uc3m | Universidad Carlos III de Madrid

Grado en Ingeniería Informática

2020-2021

Trabajo Fin de Grado

Desarrollo de un editor multiplataforma para el controlador MIDI Electrix Tweaker

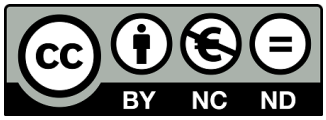
---

Jorge Marcos Chávez

Tutor

Jorge Pleite Guerra

Leganés, septiembre de 2021



RESUMEN

En este proyecto se plantea el diseño y desarrollo de un *software* editor para el Electrix Tweaker, un controlador MIDI. La aplicación permite al usuario configurar los parámetros de este *hardware* de forma sencilla a través de una interfaz gráfica. La comunicación entre el editor y el controlador se realiza utilizando el protocolo MIDI a través de USB.

El diseño de la aplicación se lleva a cabo de forma teórica atacando dos frentes: por un lado, el de ingeniería de *software*, para definir el funcionamiento interno del programa; por otro, la experiencia de usuario, que determina la estructuración de la interfaz gráfica y cómo el usuario interacciona con ella. Posteriormente, este diseño se implementa usando Java y JavaFX y se lleva a cabo un conjunto de pruebas para su verificación.

El objetivo de la aplicación es resultar de utilidad a los usuarios del Electrix Tweaker, sustituyendo el editor oficial por uno con licencia de código abierto que sea mantenible y mejorable en el tiempo por la comunidad.

Palabras clave: Controlador MIDI, editor, Java, JavaFX, interfaz gráfica, código abierto.

DEDICATORIA

A todas las personas que me introdujeron en la música y la informática, las que me motivaron (a buenas y a malas) para continuar con mis estudios, y a las que están por llegar a mi vida.

Al Lego, el Ford Racing de los cereales, el Underground 2, *Rock n Roll (Will Take You to the Mountain)*, el Midnight Club 3 y las 174 pulsaciones por minuto.

A mi familia.

ÍNDICE GENERAL

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

1. INTRODUCCIÓN

No hay duda de que la informática se ha introducido ya en todos los ámbitos de la sociedad, podemos encontrar ordenadores en prácticamente todos los puestos de trabajo en cualquier lugar del mundo. Este movimiento comenzó a finales de los años 80, cuando se introdujeron los primeros ordenadores personales [1], que permitieron realizar muchas de las tareas para las que se necesitaban varias herramientas con un único dispositivo. Por ejemplo, en contabilidad se pudieron sustituir los libros de cuentas y los cálculos manuales, haciendo el trabajo mucho más rápido y facilitando el almacenamiento y la gestión de la información [2].

Este proceso, sin embargo, no afectó a todos por igual y la transformación no tuvo lugar a la misma velocidad. En el mundo de la producción audiovisual, los requerimientos de capacidad de procesamiento eran mayores, lo que impidió sustituir el total de las herramientas usadas para esos trabajos por un ordenador: las tareas menos costosas a nivel de CPU, como secuenciar MIDI o editar audio digital de forma *offline* (no en tiempo real) se llevan pudiendo realizar desde finales de los años 90, mientras que una aplicación que pudiera sustituir totalmente un estudio de música no existió hasta los años 2000 [3].

Por otra parte, a pesar de que sustituir gran parte del *hardware* por instrumentos y efectos virtuales permitía trabajar a mayor velocidad y simplificar muchas tareas, en algunas ocasiones era contraproductivo por el cambio en la experiencia de usuario y la forma de interaccionar con ellos. Esto es evidente cuando se habla de instrumentos musicales o superficies de control: Si se sustituye un conjunto de controles físicos dedicados (que permiten grabar movimientos en tiempo real de forma cómoda y natural) por un teclado y un ratón, el nivel de flexibilidad aumenta pero la naturalidad de la interacción con el sistema disminuye [4].

Para reemplazar esta interacción del usuario se han utilizado desde entonces los controladores MIDI, unos periféricos que permiten, a través de controles físicos como ruedas y botones, modificar los parámetros de los instrumentos y efectos virtuales en el ordenador. Esto crea un puente entre la usabilidad del *hardware* y la flexibilidad del *software* que aporta un gran valor al flujo de trabajo de la producción audiovisual tanto en un estudio

Fig. 1.1. Vista general del Electrix Tweaker

como en situaciones de directo.

El Electrix Tweaker (figura 1.1) es un controlador MIDI lanzado en 2012 en una colaboración entre Livid, un fabricante de superficies de control MIDI configurables, y Mixware, los dueños del nombre Electrix, una marca de efectos digitales [5]. Dentro del mercado de este tipo de *hardware*, el Tweaker se caracteriza por tener una gran variedad de controles distintos en una disposición orientada al *DJing* (selección y reproducción de música pregrabada para ser escuchada por una audiencia) y *controllerism* (uso de controladores MIDI para generar un contenido audiovisual en directo), y por ser totalmente configurable: aunque está diseñado para un cierto tipo de flujos de trabajo, se presta a adaptarse a cualquier tarea dentro del control a través de MIDI.

1.1. Motivación

El Electrix Tweaker cuenta con una gran variedad de parámetros de configuración distintos. Acceder a ellos y modificarlos a través de un *software* editor que se encargue de mostrarlos en una interfaz gráfica de forma intuitiva es fundamental para su uso efectivo: sin el editor, se pierde gran parte de la funcionalidad y la usabilidad del *hardware*. En este caso, el Tweaker cuenta con un editor oficial, mostrado en la figura 1.2, que permite:

- Modificar qué mensajes MIDI manda cada control.
- Cambiar los colores de las luces LED y su modo de funcionamiento.
- Guardar la configuración en un fichero local.
- Cargar una configuración desde un fichero y aplicarla al *hardware*.
- Modificar parámetros globales y establecer los parámetros actuales como configuración por defecto.

Este editor oficial se distribuye sin coste en la página oficial de Electrix, y se incluyó en un CD empaquetado con el *hardware* en la caja, en su versión 1.0. Durante la vida útil del controlador se ha actualizado el editor una única vez, a la versión 1.0.1, a fecha de octubre de 2014 [6].

El problema que ocurre con frecuencia, incluyendo el presente caso, es que la esperanza de vida de sintetizadores, controladores y módulos *hardware* es generalmente mayor que la de los editores e instrumentos virtuales *software*. Mientras que el funcionamiento de un editor depende de que el ordenador pueda ejecutar el código físicamente y el sistema operativo sea compatible con él, un controlador MIDI tan genérico como el Tweaker puede funcionar hasta que los propios materiales y componentes se degraden. En general, los sintetizadores y dispositivos MIDI, si tienen una construcción sólida y se mantienen correctamente, pueden llegar a durar décadas [7], ya que están pensados generalmente para su uso intensivo en situaciones de directo, donde se exponen a un mayor desgaste.

En este caso, el Editor oficial del Electrix Tweaker fue actualizado por última vez en 2014 [6], y por ello sufre de varios problemas de compatibilidad con los sistemas operativos de 2021:

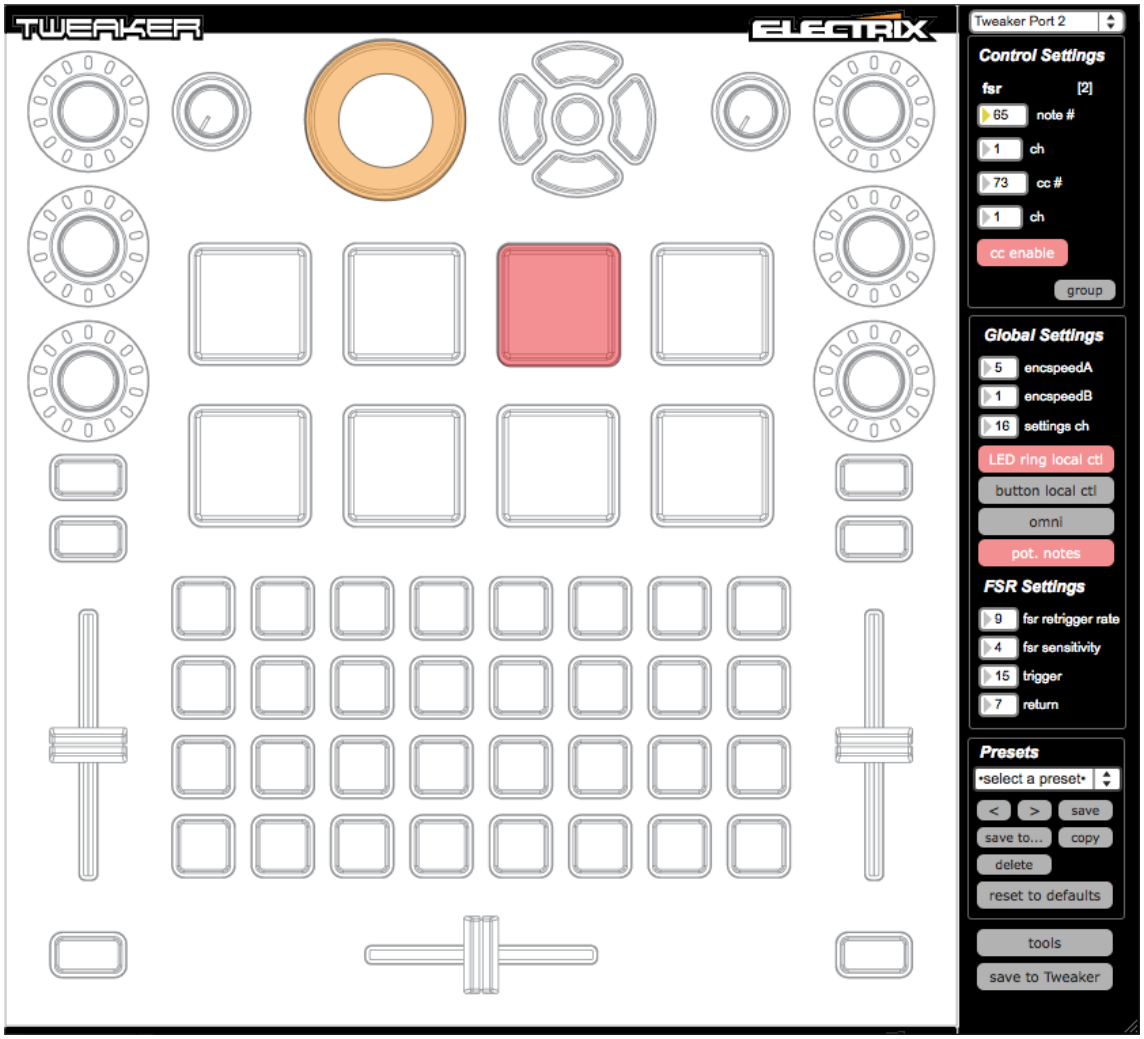


Fig. 1.2. Vista de la aplicación editor original

- En macOS 11 el Editor no está certificado y si se intenta ejecutar, el sistema operativo bloquea la operación. Se puede evitar este comportamiento temporalmente cambiando la configuración de seguridad, pero es una solución poco óptima si lo que se requiere es que el editor sea fácilmente accesible.
- En Windows 10 el Editor se ejecuta pero no reconoce el controlador aunque esté correctamente conectado. El sistema operativo sí reconoce el dispositivo MIDI, y funciona con otros programas (Reaper, FL Studio y Mixxx).
- En Linux el Editor no funciona. No hay versiones para ninguna distribución de este sistema operativo.

Una posible solución para este problema es liberar el código fuente del Editor. El *software* libre es aquel cuyos usuarios tienen la posibilidad de ejecutar, distribuir y modificar su código, entre otras libertades [8]. En el caso de un editor para un controlador MIDI que se distribuye sin coste alguno y cuya funcionalidad está totalmente documentada, si el desarrollador original no puede continuar dando soporte a la aplicación no hay razón de peso para no publicar el código original con una licencia que permita a los usuarios, al menos, solucionar problemas de compatibilidad. En el caso del Editor original del Tweaker, el código podría ser analizado y depurado para corregir el fallo que le impide reconocer el controlador.

1.2. Objetivos

El objetivo de este trabajo es, por una parte, solucionar los problemas planteados en la sección 1.1, desarrollando una aplicación que sea:

1. Útil para los usuarios del Electrix Tweaker, es decir, que funcione como un sustituto del editor oficial, basándose en él y permitiendo editar como mínimo las configuraciones más usadas.
2. *Software* libre, para que la comunidad pueda seguir dando soporte a la aplicación en caso de que sea necesario sin que toda la responsabilidad de mantenerla recaiga sobre unas pocas personas o una única entidad, y que el esfuerzo de implementar

todas las funcionalidades que ofrece el editor original sea potencialmente colaborativo.

3. Multiplataforma. Ya que el controlador es independiente del sistema operativo al que se conecte, tiene sentido que el editor también funcione en, al menos, los tres sistemas operativos principales (Windows, macOS, y Linux).

4. Fácil de desarrollar y estable, para conseguir al menos un prototipo de la aplicación que funcione correctamente pero que también sea extensible a largo plazo.

5. Una mejora sobre el Editor original, por las características presentadas anteriormente y por la inclusión de funcionalidades de las que no disponía el antiguo Editor.

Un objetivo secundario del proyecto es la exploración de las soluciones de *software* actuales que permiten diseñar y programar una aplicación con interfaz gráfica con capacidades para enviar y recibir información a través del protocolo MIDI. Dado que tiene limitados casos de uso y que la demanda de este tipo de aplicaciones es reducida, las opciones disponibles son menores, lo que permite realizar un análisis rápido.

1.3. Glosario

1.3.1. Acrónimos

- **API**: *Application Programming Interface*, o Interfaz de Programación de Aplicaciones.
- **CC**: *Control Change*.
- **CD**: *Compact Disk*, o Disco Compacto.
- **CPU**: *Central Processing Unit*, o Unidad Central de Procesamiento.
- **CSS**: *Cascading Style Sheet*, u Hoja de Estilo en Cascada.
- **DIN**: *Deutsches Institut für Normung*, u Organización Nacional de Estándares de Alemania.
- **DJ**: *Disc Jockey*, o pinchadiscos.

- **GPL**: GNU General Public License, o Licencia Pública General de GNU.
- **IRPF**: Impuesto sobre la Renta de las Personas Físicas.
- **JDK**: *Java Development Kit*, o Kit de Desarrollo de Java.
- **JRE**: *Java Runtime Environment*, o Entorno de Ejecución de Java.
- **JSON**: *JavaScript Object Notation*, o Notación de Objeto de JavaScript.
- **JVM**: *Java Virtual Machine*, o Máquina Virtual de Java.
- **LED**: *Light-Emitting Diode*, o Diodo Emisor de Luz.
- **MIDI**: *Musical Instrument Digital Interface*, o Interfaz Digital para Instrumentos Musicales.
- **MMA**: *Midi Manufacturers Association*, o Asociación de Fabricantes de dispositivos MIDI.
- **RAM**: *Random Access Memory*, o Memoria de Acceso Aleatorio.
- **RGB**: *Red, Green & Blue*, o (luz) Roja, Verde y Azul (multicolor).
- **UI**: *User Interface*, o Interfaz de Usuario.
- **USB**: *Universal Serial Bus*, o Bus Universal en Serie.
- **UX**: *User Experience*, o Experiencia de Usuario.

1.3.2. Definiciones

- **Biblioteca** (de código): Conjunto de código con funciones y estructuras de datos que proporcionan una serie de funcionalidades, con el objetivo de encapsularlo y permitir su reutilización para facilitar y hacer más rápido el desarrollo de un programa.
- **Bit**: Unidad de información básica en un ordenador. Puede tener un valor de 0 o de 1.

- ***Byte***: Grupo de 8 bits. Agrupación básica de bits usada en gran parte del *hardware* y *software* informático y de telecomunicaciones.

- **Certificar** (*software*): En los sistemas operativos macOS o Windows, firmar criptográficamente una aplicación usando certificados digitales para que, en el momento de ejecutarla, el sistema operativo pueda validar su origen. En macOS, la capa de seguridad que verifica la firma hace más complicado el ejecutar una aplicación no certificada que en Windows.

- **Codificador rotativo**: Dispositivo electromecánico que convierte la posición angular de un eje en una señal analógica o digital [9]. A efectos de un controlador MIDI, el nombre "codificador" se usa para referirse a un codificador rotativo incremental, es decir, que sólo proporciona información sobre cuánto se está girando el eje y hacia qué dirección, pero no su posición absoluta.

- **Compilación cruzada**: Generación de un binario ejecutable en una plataforma que no es aquella en la que se realiza este proceso. Compilación para otras plataformas o sistemas operativos.

- **Control**: En un sintetizador, superficie de control MIDI o similar, componente físico que permite la modificación de un parámetro del instrumento o dispositivo receptor, como por ejemplo un potenciómetro o un botón.

- ***Controllerism***: Disciplina de la presentación artística en la que se utiliza uno o más controladores MIDI para generar un contenido audiovisual o musical en directo.

- ***DJing***: Actividad en la que una persona selecciona y reproduce música pregrabada para ser escuchada por una audiencia. En ocasiones, parte del equipo técnico usado está formado por un ordenador y uno o varios controladores MIDI.

- **Experiencia de usuario**: Cómo un usuario interactúa con un sistema. El diseño de experiencia de usuario trata de planificar dicha interacción, haciendo los cambios pertinentes en el sistema para conseguir los objetivos deseados.

- ***Framework***: Conjunto de herramientas y bibliotecas que proporcionan un esqueleto o base sobre la que desarrollar una aplicación, facilitando y acelerando el proceso pero a veces restringiendo la posibilidad de tomar decisiones de diseño alternativas.

- **Mensaje MIDI de nota**: Mensaje MIDI enviado por un controlador cuando se interactúa con una nota. Puede ser de tipo *note on* (presionar) o *note off* (soltar). Como parámetros del mensaje contiene el número de nota y la fuerza de presionado (0 si se trata de un *note off*), ambos valores con un rango entre 0 y 127.

- **Mensaje MIDI de control**: Similares a los de nota, pero describen el cambio de un parámetro del instrumento musical. Sus parámetros son el número de control y su valor, ambos con el mismo rango entre 0 y 127.

- ***Offline***: En el contexto del procesamiento de vídeo o de audio, una operación que no es posible realizar en tiempo real, sino que por su alto coste computacional el tiempo de procesado es mayor que la duración del contenido resultado, impidiendo su procesado y reproducción simultánea.

- ***Plug & Play***: Dicho de un periférico, que no requiere de la instalación o configuración al ser conectado a un ordenador. Suelen utilizar controladores ya incluidos en el sistema operativo.

- **Potenciómetro rotativo**: Similar al codificador rotativo, pero en este caso la información que proporciona es la posición absoluta del eje como un entero en un rango. Tienen un mínimo y un máximo, con topes físicos que impiden al control superar ese rango.

- **Potenciómetro lineal**: Similar al potenciómetro rotativo, pero en este caso el movimiento no es angular, sino que se desarrolla en un segmento recto.

- ***Scripting***: Estilo de programación que utiliza lenguajes no compilados, sino que requieren un ejecutable (en ocasiones llamado máquina virtual) que interpreta el código y ejecuta las operaciones necesarias en tiempo real.

- **Sintetizador**: Instrumento musical que genera señales de audio de forma electrónica. Puede o bien disponer de un teclado para introducir las notas que se quieren generar, o tener un puerto de entrada MIDI para recibir esas señales de otro sintetizador o de un controlador MIDI.

- **SysEx**: Mensajes de longitud variable que permiten a los fabricantes de equipo

compatible con MIDI definir sus propios tipos de mensajes, como volcados de información o configuración de parámetros no especificados [10].

1.4. Estructura del documento

Este documento se estructura en 14 capítulos explicados a continuación:

1. **Introducción**: Breve explicación del contexto y la problemática tratada en este proyecto.
2. **Estado del arte**: Estudio de las tecnologías usadas en el contexto del problema y de las principales soluciones competidoras.
3. **Análisis del problema**: Estudio detallado de la problemática y definición de la solución propuesta.
4. **Especificación de requisitos**: Definición de requisitos funcionales y no funcionales.
5. **Diseño de casos de uso**: Definición de casos de uso y su trazabilidad con los requisitos.
6. **Arquitectura del sistema**: Descripción del sistema de forma teórica.
7. **Implementación y diseño de UX**: Detalle del entorno de desarrollo y del diseño de la interfaz gráfica.
8. **Pruebas**: Definición de las pruebas del sistema y sus resultados.
9. **Marco regulador**: Análisis de aspectos legales relevantes.
10. **Impacto socioeconómico**: Análisis del alcance social y económico del proyecto.
11. **Planificación**: Descripción de la estructuración del proyecto a nivel organizativo.
12. **Análisis económico**: Desglose del presupuesto del proyecto y análisis de riesgos y beneficios.
13. **Conclusiones y trabajo futuro**: Veredicto del proyecto, y posibles cambios y mejoras.

14. **Extended abstract**: Descripción breve del proyecto en inglés.

2. ESTADO DEL ARTE

Para entender la situación en la que se encuentra el desarrollo del proyecto a nivel técnico, es imprescindible conocer:

- De qué trata el protocolo MIDI.
- Cómo se puede utilizar para interaccionar con dispositivos externos con un lenguaje de programación usando bibliotecas.
- Las opciones existentes para desarrollar una aplicación multiplataforma con interfaz de usuario gráfica.

También resulta interesante investigar el comportamiento y funcionalidades de aplicaciones alternativas a la que se quiere desarrollar, para partir de una base comparable a las soluciones actuales y planificar el proyecto para posibles ampliaciones y futuras mejoras.

2.1. Protocolo MIDI

Un controlador MIDI es una pieza de *hardware* que, al igual que un teclado o una pantalla, actúa de periférico para un instrumento electrónico, módulo MIDI u ordenador. Cuenta con varios tipos de controles (botones, codificadores rotativos, potenciómetros, etcétera) y elementos visuales (normalmente luces, y en ciertas ocasiones el movimiento automático de algunos controles) que proporcionan una respuesta a las acciones del usuario.

Siendo un periférico, se limita a comunicar al ordenador o módulo MIDI las órdenes indicadas por el usuario: la acción de pulsar un botón con cierta fuerza o la de desplazar un codificador rotativo en sentido antihorario, por ejemplo. El uso más común del MIDI es el enlace de un generador de sonidos, ya sea un ordenador o un módulo *hardware*, con un controlador de tipo piano que permita al usuario tocar un instrumento.

Este intercambio se lleva a cabo usando el protocolo MIDI, un estándar de comunicación que describe notas y parámetros musicales mediante *bytes*, y que se puede realizar

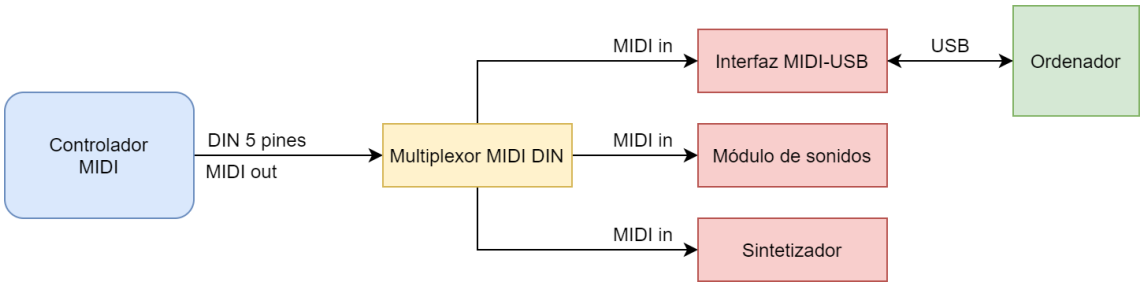


Fig. 2.1. Esquema de ejemplo de conexiones entre un controlador y varios dispositivos con capacidades MIDI

Status:	1	0	0	1	C	C	C	C
Data 1:	0	T	T	T	T	T	T	T
Data 2:	0	V	V	V	V	V	V	V

Fig. 2.2. Estructura del mensaje MIDI de tipo *note on*

a través de cables con conectores tipo DIN de 5 pines, cables USB o incluso tecnologías *wireless* como *Bluetooth* o protocolos privativos [11]. Puede ser unidireccional o bidireccional, y se puede aprovechar tanto para mandar las órdenes del usuario al generador de sonido como para devolver los cambios de la interfaz desde el generador hacia el controlador (encendiéndose una luz al pulsar un botón, denotando que se ha activado una función concreta del generador de sonido).

Los mensajes MIDI están formados por 3 *bytes*. Cada uno tiene un primer bit que identifica si se trata de un *byte* de estado o de datos, y el resto de bits describen su contenido [12]. Dos ejemplos de mensajes MIDI son el *note on*, usado frecuentemente para comunicar que se está pulsando una tecla, y el *note off*, cuando se suelta. Sus estructuras en *bytes* se pueden ver en las figuras 2.2 y 2.3 [13], donde T son los bits que componen el tono (número de nota) y V los bits de la velocidad (fuerza).

Los mensajes SysEx son un conjunto de N *bytes*. Se trata de un tipo de mensaje ge-

Status:	1	0	0	0	C	C	C	C
Data 1:	0	T	T	T	T	T	T	T
Data 2:	0	V	V	V	V	V	V	V

Fig. 2.3. Estructura del mensaje MIDI de tipo *note off*

nérico donde la longitud y la interpretación del contenido los define el fabricante del *hardware*.

Al ser un protocolo pensado para ser simple y ligero, lo más genérico posible, y funcionar en una gran cantidad de dispositivos diferentes, se puede usar en aplicaciones ajenas al ámbito de la producción musical o del control de instrumentos virtuales como el control de dispositivos electrónicos genéricos, siendo una alternativa económica y simple a otros sistemas [14]. Además, la mayoría de controladores MIDI son *plug & play* y cumplen con la especificación MIDI USB que les permite funcionar con la mayoría de los sistemas operativos cuando se conectan a un ordenador sin necesidad de *drivers* específicos [15].

La especificación MIDI consta de dos versiones principales:

- **Versión 1.0**: Especificada en 1996 a pesar de que el MIDI llevaba en uso más de una década. Es el estándar que se sigue usando ahora mismo, y cuenta con una gran compatibilidad, pudiendo comunicar dispositivos con más de 30 años de diferencia de fechas de fabricación. Después de su publicación en 1983 se han añadido extensiones, tanto por parte de la MMA como de los propios fabricantes de *hardware*, como el *General MIDI* y el código de tiempo.

  Este estándar 1.0 también proporciona tipos de mensajes genéricos para que tanto los fabricantes como los usuarios de los dispositivos tengan la opción de expandir las capacidades del MIDI según les sea necesario.

- **Versión 2.0**: Presentada en enero de 2020, mantiene la compatibilidad con la versión anterior pero lleva a cabo una modernización del protocolo, extendiendo la resolución de los parámetros que permite modular y abstrayendo el protocolo de su capa física (ya no queda ligado principalmente al conector DIN, sino que permite implementaciones sobre otras capas, en redes y en sistemas sin cables).

  Ya existen algunos controladores MIDI que implementan el MIDI 2.0, pero la gran mayoría del *hardware* y *software* no es aún compatible [16], aunque la mayoría de aplicaciones y parte del equipo físico podrán actualizarse para integrar este nuevo protocolo.

2.2. Bibliotecas MIDI

Para llevar a cabo la comunicación entre el Electrix Tweaker y la aplicación editor se debe tener la capacidad de enviar y recibir mensajes MIDI y SysEx. Para esto, existen diferentes bibliotecas para distintos lenguajes de programación que se enumeran a continuación.

En general, se pueden establecer dos fases del funcionamiento de la comunicación con un dispositivo MIDI, indicadas en la figura 2.4:

1. **Fase de conexión**: Se obtiene el conjunto de dispositivos MIDI que hay disponibles (conectados al ordenador, reconocidos por el sistema operativo), se selecciona uno o varios con los que se quiere interaccionar, y se fijan como activos.

2. **Fase de comunicación**: Se envían y/o reciben mensajes MIDI. Estas acciones se realizan normalmente a petición del usuario, contando con una cola de mensajes recibidos para mantener la asincronía de la operación. En el caso de que el dispositivo previamente activo salga de ese estado (desconectado físicamente, o no disponible por cualquier razón) se deberá buscar un nuevo dispositivo antes de enviar o recibir mensajes de nuevo.

Cómo se realicen estas dos fases y el proceso en detalle de cada una de ellas depende de la biblioteca en uso. Por ejemplo, la decisión de qué dispositivo MIDI se marca como activo es, en algunos casos, relegada al usuario, que debe buscar y luego seleccionar explícitamente; en otros se realiza automáticamente, buscando y seleccionando todos los dispositivos disponibles.

2.2.1. RtMidi

RtMidi es un conjunto de clases de C++ que proveen una API común en Windows, macOS y Linux para interaccionar con dispositivos MIDI. Se trata de una capa de abstracción por encima de las APIs nativas de interacción con este protocolo de cada sistema operativo [17].

A pesar de ser una biblioteca para C++, tiene adaptadores para C, Python y Go, entre

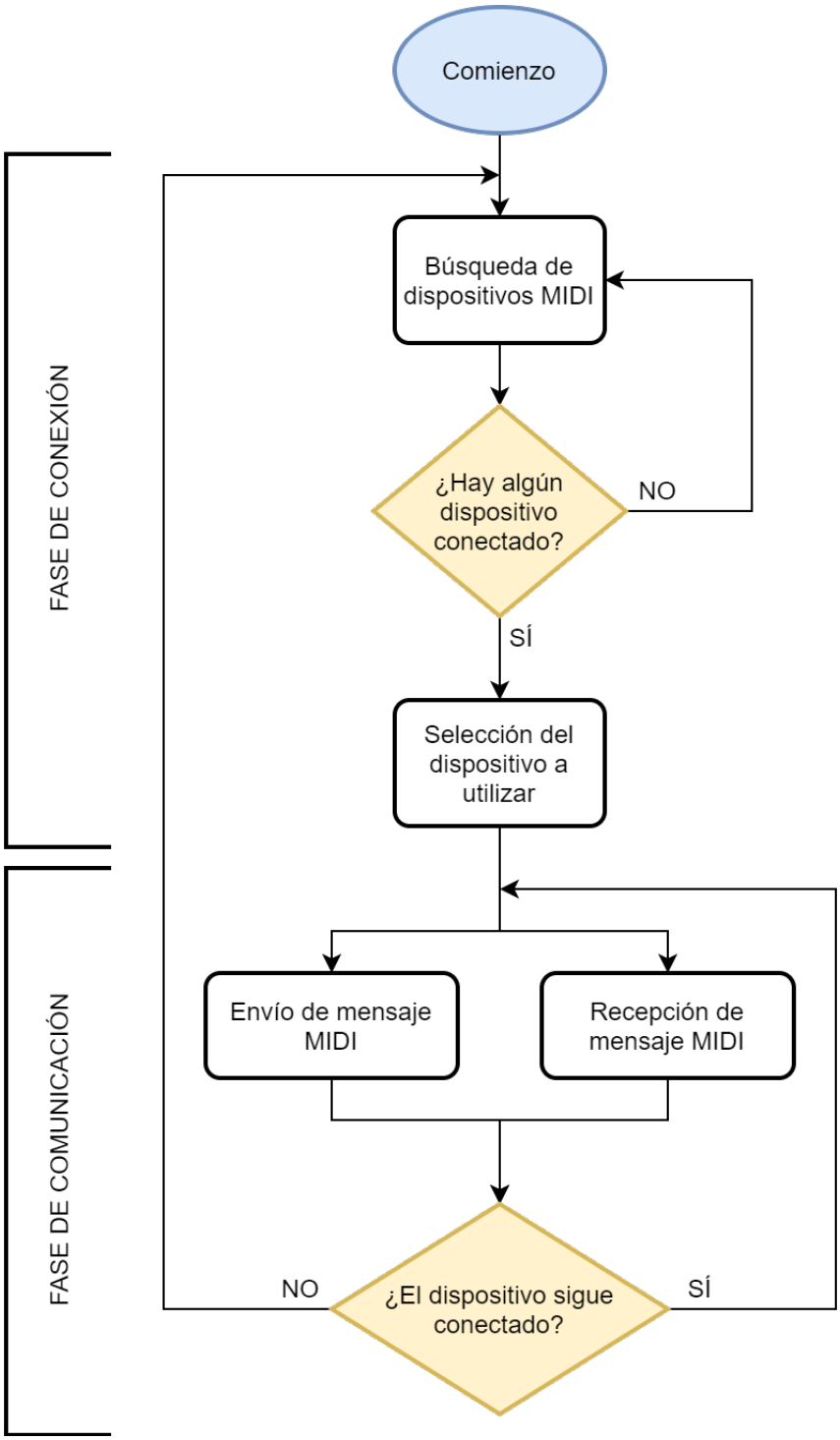


Fig. 2.4. Esquema del proceso de funcionamiento general de la interacción entre aplicación y dispositivo MIDI

otros [18] [19], aunque en algunos casos requiere de su instalación en el sistema como biblioteca dinámica. Esto supone varios problemas:

- Si se usa con un lenguaje compilado, requiere trabajo añadido por parte del programador para integrar la biblioteca en el entorno de compilación y montado del programa.
- Si se usa con un lenguaje interpretado, elimina portabilidad de la aplicación, ya que pasa a depender tanto del entorno de ejecución del lenguaje como de la instalación de la biblioteca dinámica en el sistema, y requiere controlar además la versión de la biblioteca, que puede no ser la misma en distintos sistemas operativos.
- Si se usa con un adaptador, se trata de una pieza de código extra que depende de un tercero y que puede ser una fuente de errores durante el desarrollo.

Por otra parte, sus ventajas son:

- Alto rendimiento ya que está escrita en C++, aunque esto es beneficioso únicamente si necesitamos hacer un uso intensivo y preciso del protocolo MIDI, como por ejemplo controlar decenas de dispositivos en tiempo real y modificar cientos de parámetros al mismo tiempo.
- Abundante documentación, por el hecho de ser una de las bibliotecas más usadas para implementar funcionalidades MIDI.

2.2.2. Java Sound API

Java Sound API es un *framework* que permite el acceso de bajo nivel a grabaciones y reproducción de audio, secuenciado y envío y recepción de MIDI, entre otras funciones [20]. Forma parte de las bibliotecas que se empaquetan en las distribuciones del entorno de ejecución de Java, incluyendo OpenJDK.

El hecho de ser parte de Java define unas ventajas claras:

- La portabilidad y la condición de multiplataforma se mantienen al estar empaquetada por defecto en los principales JDK.

- Es fácil de integrar en el entorno de desarrollo y de incluir en la generación de la aplicación final.
- Cuenta con documentación tanto directamente desde Oracle como de terceras partes muy exhaustiva.

La mayor inconveniencia es que imposibilita su uso con ningún otro lenguaje de programación (que no se ejecute sobre JVM), por lo que no es una opción si no se quiere o puede usar Java.

2.2.3. Web MIDI API

Define una API que permite a las aplicaciones web acceder y comunicarse con los dispositivos MIDI conectados en el ordenador cliente [21]. Es una tecnología web, implementada exclusivamente en Chromium (y sus derivados, incluyendo Edge, Opera y el WebView de Android) en 2021 [22]. Se trata de una solución altamente portable ya que puede ejecutarse en cualquier ordenador que disponga de alguno de estos navegadores web, pero a la vez significa una restricción de uso y de entorno de desarrollo muy estricta (un navegador, Electron, o Node.js).

El lenguaje de programación utilizado para trabajar con Web MIDI API es JavaScript.

2.3. Herramientas multiplataforma para el desarrollo de interfaces de usuario

Existen diversas soluciones que permiten diseñar e implementar una interfaz gráfica, cada una con sus ventajas y sus inconvenientes dadas las circunstancias de este proyecto. Se excluyen a continuación las opciones que no ofrecen la posibilidad de generar una aplicación que funcione en varios sistemas operativos.

2.3.1. Qt

Qt es un *framework* para el desarrollo de aplicaciones multiplataforma, escrito en C++ y con licencias GPL (disponible de forma gratuita) y comercial (con un coste monetario, pero elimina las restricciones de distribución de GPL: el código que se genera usando una herramienta con dicha licencia debe usar esta misma). Cuenta con un editor propio, Qt

Creator, que gestiona todo el proceso de compilación. La primera versión fue lanzada en 1995 [23], por lo que su uso está probado y documentado.

Aunque otros lenguajes de programación cuentan con *bindings* de Qt, el hecho de que emplee C++ a la hora de desarrollar la aplicación puede suponer un problema en el momento de compilar el código para distintas plataformas, ya que la compilación cruzada entre cualquier plataforma no forma parte del *framework* [24].

2.3.2. Electron

Basado en tecnologías web (HTML, CSS y JavaScript), es relativamente nuevo (la primera versión pública data de 2017 aunque estuvo en desarrollo privado en GitHub, destinado a Atom) y permite reducir los tiempos de desarrollo e integración al utilizar tecnologías que, desde su origen, están pensadas para funcionar en varias plataformas usando el mismo código.

Esta portabilidad y conveniencia al desarrollador vienen dadas a costa del consumo de recursos, tanto de memoria RAM como de almacenamiento en disco y uso de procesador, ya que requiere la ejecución de una copia exclusiva del navegador web Chromium y una instancia de Node.js por cada programa que utilice este *framework* [25].

2.3.3. JavaFX

JavaFX es un *framework* multiplataforma para crear aplicaciones con interfaz de usuario gráfica usando Java. Inicialmente desarrollado por SeeBeyond como un lenguaje de *scripting* para programar aplicaciones con interfaz gráfica de forma sencilla, tras una serie de adquisiciones entre compañías terminó siendo propiedad de Oracle, y fue presentado en 2010 [26]. A partir del lanzamiento de OpenJDK 11 se liberó como código libre.

Dado que se trata de un conjunto de bibliotecas para Java, JavaFX es igual de portable y multiplataforma.

2.3.4. Go con paquetes de terceros

Go es un lenguaje de programación de código abierto orientado hacia la programación de sistemas lanzado al público por primera vez en 2009. Está inspirado por el lenguaje C pero fue diseñado desde cero para resolver problemas de ingeniería de *software* que sufren otros lenguajes como Java y C++ a la hora de ser utilizados en grandes proyectos [27]. Es compilado pero permite realizar compilación cruzada de una forma muy sencilla [28] y cuenta con soporte para los sistemas operativos y arquitecturas más usadas, por lo que es una buena opción para desarrollar código multiplataforma.

El desarrollo de interfaces de usuario gráficas en Go es más complicado que en las opciones anteriores, ya que no cuenta con ninguna solución oficial. Existen multitud de paquetes que enlazan con GTK, Qt y varios *frameworks* basados en tecnologías web, pero su soporte para multiplataforma y la calidad de la documentación varían al ser proyectos pequeños y mantenidos por la comunidad [29].

2.3.5. Python con Tkinter

Se trata de un lenguaje de *scripting* interpretado, con soporte para un gran número de sistemas operativos y arquitecturas diferentes, y una sintaxis clara y potente. Siendo interpretado, requiere que el sistema donde se vaya a ejecutar tenga el intérprete de *Python* instalado previamente, con las versiones correspondientes (y las dependencias descargadas).

Al igual que Go, no tiene soporte oficial para ningún *frawework* de interfaces de usuario gráficas, pero por defecto incluye una interfaz que permite enlazar Python con Tkinter, instalado en todos sus binarios [30].

2.4. Aplicaciones competidoras

Para los propósitos de este análisis, se puede hacer una distinción entre editores genéricos (compatibles con cualquier dispositivo MIDI) y específicos (orientados a su uso con el Electrix Tweaker).

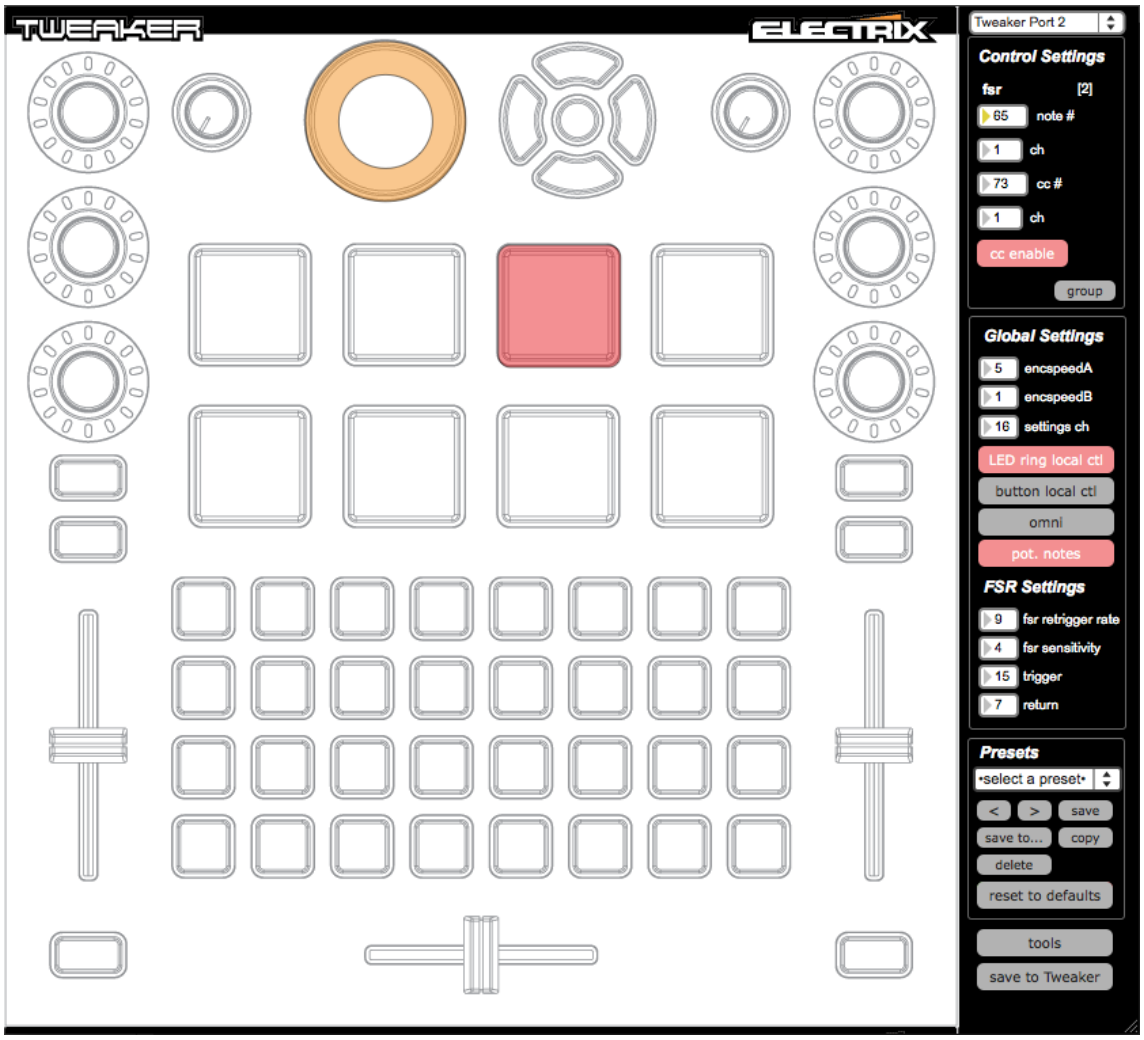


Fig. 2.5. Vista de la aplicación editor original

2.4.1. Editor oficial

Es una aplicación diseñada específicamente para acompañar al Electrix Tweaker. Distribuida oficialmente con el controlador MIDI en un CD en la caja junto al *hardware* y disponible como descarga en línea.

Como se puede ver en la figura 2.5, cuenta con una interfaz gráfica con una vista cenital del Tweaker, que permite seleccionar con el ratón los controles que se desean editar. En el ejemplo de la figura, el *pad* 3 (primera fila, tercera columna) está seleccionado (apreciable ya que se encuentra resaltado en rojo) y sus parámetros se pueden editar cambiando los valores de la caja *Control settings* en la sección derecha de la pantalla.

Las funcionalidades principales de esta aplicación son:

- Editar todos los parámetros del Electrix Tweaker, como se detalla en el manual [6].
- Editar la configuración global del controlador. La configuración se sincroniza cada vez que se hace un cambio en la ventana.
- Establecer la configuración actual del controlador como inicial, activada al encenderlo. Por defecto, cualquier cambio realizado en el Tweaker se sustituye por la configuración por defecto tras reiniciarlo.
- Cargar la configuración actual del *hardware* al editor, haciéndole una petición al controlador para que envíe su estado en ese momento.
- Guardar las configuraciones hechas en el editor en archivos locales del ordenador. En lugar de enviarlas al Tweaker, se almacenan para poder cambiar todos los parámetros más rápidamente, y alternar entre configuraciones.
- Cargar una configuración desde un archivo local al editor y al *hardware*.

2.4.2. Ctrlr

Una aplicación de código abierto que permite a los usuarios crear sus propias interfaces gráficas para controlar *hardware* MIDI [31].

Es genérica ya que, dependiendo de la interfaz que cargue el usuario, el *software* puede interaccionar con unos dispositivos u otros. Por ejemplo, en la figura 2.6 se puede observar el panel de edición del Oberheim Matrix 1000 en Ctrlr.

Para el diseño de estos paneles cuenta con un sistema de arrastrar y soltar componentes predefinidos sobre una cuadrícula, que luego el usuario puede personalizar tanto estéticamente como funcionalmente definiendo los mensajes MIDI que mandará cada uno. Esto permite obtener un editor en poco tiempo, pero a su vez limita sus capacidades a las ofrecidas por el programa.

Para intentar paliar en la medida de lo posible estas limitaciones, este editor permite crear componentes dinámicos en los paneles usando el lenguaje de *scripting* Lua.

Las funcionalidades que ofrece como editor dependen del panel que esté en uso, pero por lo general todas dependen de:

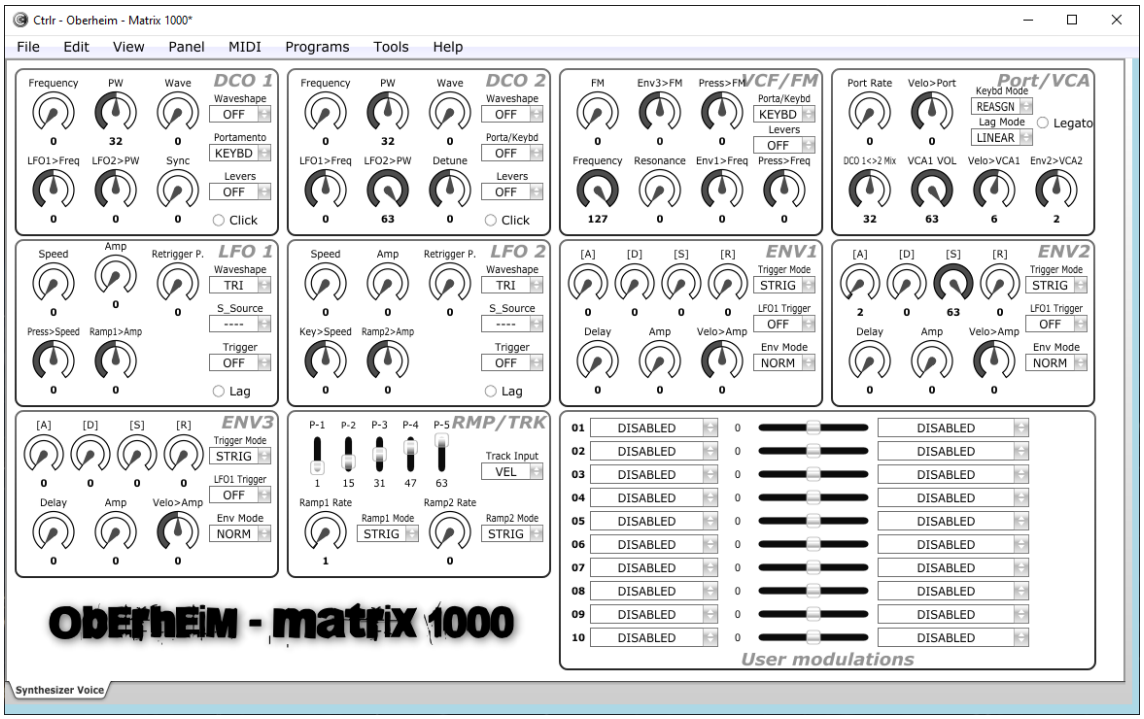


Fig. 2.6. Vista del panel del Oberheim Matrix 1000 en Ctrlr

- Enviar mensajes MIDI y SysEx interaccionando con la interfaz gráfica.
- Recibir mensajes MIDI y SysEx, y modificar los componentes de la interfaz según su contenido.

Partiendo de estas capacidades de comunicación MIDI, y combinándolas con el diseño de interfaz y el *scripting* con Lua, el usuario puede implementar la edición de parámetros, de la configuración global, o el volcado de información, por ejemplo, aunque siempre contando con las limitaciones intrínsecas de la implementación de esta aplicación: ciertos comportamientos de mayor complejidad no son posibles.

3. ANÁLISIS DEL PROBLEMA

A continuación se realiza un estudio de la problemática planteada, analizando la parte de *hardware* que contempla el proyecto, y se propone una solución de *software*.

3.1. Análisis del *hardware*

Para gran parte de este análisis se utiliza el manual de usuario del Tweaker (*Tweaker user manual*), descargable desde la página de Electrix [6].

El Electrix Tweaker es un controlador MIDI que consta de botones, *pads* sensibles a la presión, potenciómetros (tanto rotativos como lineales) y codificadores rotativos, todos ellos visibles en la figura 3.1, con conexión USB tipo Standard-B para comunicarse con un ordenador, y conexiones tipo DIN para comunicarse con *hardware* externo como módulos de sonido o sintetizadores (mostrados en la figura 3.2). La gran mayoría de controles tienen un juego de luces LED, monocromáticas o RGB, que responden a acciones del propio controlador o a ciertos mensajes MIDI que recibe.

En la figura 3.3 se indica el número que identifica cada control, con el siguiente código de colores:

- En color amarillo los botones.
- En color verde los potenciómetros.
- En color azul los codificadores rotativos.
- En color rojo los *pads*.

La interacción con cada control envía mensajes MIDI de distintas características:

- Los **botones** pueden enviar mensajes tipo nota, siendo *note on* cuando se pulsa y *note off* cuando se suelta, o tipo control, con un valor de 127 al pulsar y de 0 al soltar.

Fig. 3.1. Vista de los controles del Electrix Tweaker

Fig. 3.2. Conexiones del Electrix Tweaker

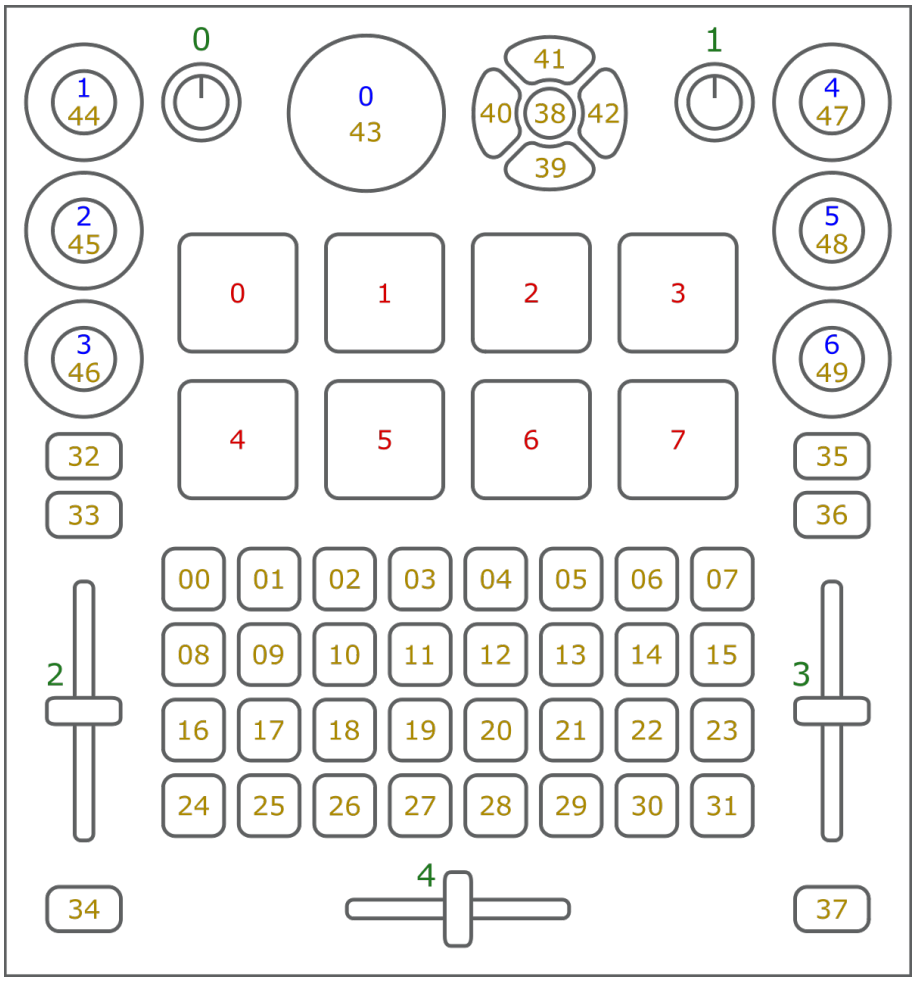


Fig. 3.3. Esquema de los controles del Electrix Tweaker

- Los **potenciómetros**, tanto los rotativos como los lineales, son controles absolutos, es decir, son un mapeo entre la posición del control y el rango de valores de un mensaje de tipo control de 0 a 127. El potenciómetro manda un mensaje sólo cuando el usuario cambia su posición: por ejemplo, si el *crossfader* está en su posición mínima hacia la izquierda (en el 0) y se desplaza hasta su posición máxima a la derecha (el 127), el controlador habrá mandado todos los mensajes MIDI de control al número correspondiente con unos valores entre el 1 y el 127 incluidos, en orden ascendente. El movimiento contrario mandará los mensajes entre el 126 y el 0, en orden descendente.

  Por otra parte, cada potenciómetro tiene un tope central que tienen la la capacidad de enviar mensajes de tipo nota, con un valor de 0 cuando el control alcanza este tope, 127 cuando abandona el centro incrementando su valor (rotación en sentido horario, por ejemplo), y 64 cuando lo abandona decrementando su valor (rotación en sentido antihorario, por ejemplo).

- Los **codificadores rotativos** funcionan de forma muy similar a los potenciómetros en el caso de este controlador, a excepción de uno de ellos, de mayor diámetro, que envía mensajes de control relativos (un valor de 127 si se mueve en el sentido horario, y 1 en el antihorario).

  Todos los codificadores rotativos cuentan con un botón, accesible pulsando la rueda de selección.

- Los ***pads***, o superficies sensibles a la presión, mandan dos mensajes distintos. En el instante en el que se presionan (impacto), se manda un mensaje tipo nota cuyo valor de velocidad es proporcional a la fuerza con la que se presiona el *pad*. Durante el tiempo que se mantenga la presión en la superficie hasta el momento en que se suelte, se mandan mensajes de tipo control cada cierto tiempo, cuyo valor es la presión en ese momento. Esta presión puede cambiar, por lo que durante el tiempo que se mantiene el *pad* presionado, se puede utilizar la fuerza aplicada como sustituto de un potenciómetro o un codificador.

Estos comportamientos se resumen en la tabla 3.1.

Los mensajes que envía cada control, los que deben recibir las luces para encenderse

Elemento	**Función**	**Tipo de mensaje**	**Valores**
Botón	Pulsar	Nota o control	127
Botón	Soltar	Nota o control	0
Potenciómetro	Rotar (+)	Control	de 1 a 127
Potenciómetro	Rotar (-)	Control	de 0 a 126
Potenciómetro	Alcanzar tope	Nota	0
Potenciómetro	Rebasar tope (+)	Nota	127
Potenciómetro	Rebasar tope (-)	Nota	64
Codificador (abs.)	Rotar (+)	Control	de 1 a 127
Codificador (abs.)	Rotar (-)	Control	de 0 a 126
Codificador (rel.)	Rotar (+)	Control	127
Codificador (rel.)	Rotar (-)	Control	1
Pad	Pulsar	Nota	de 1 a 127
Pad	Mantener	Control	de 1 a 127
Pad	Soltar	Nota	0

TABLA 3.1. FUNCIONALIDADES DE CADA CONTROL DEL ELECTRIX TWEAKER

o cambiar de color y la mayoría de parámetros son configurables enviando algún tipo de mensaje MIDI o cadenas SysEx. Esto se detalla en la tabla 3.2.

Para la gran parte de los parámetros se requieren mensajes SysEx, que para el caso del Electrix Tweaker, deben tener una estructura común, representada en la tabla 3.3 asumiendo una longitud de N *bytes*.

3.2. Definición del problema de ingeniería

El problema planteado es el diseño de un sistema que permita configurar los parámetros internos del controlador MIDI *hardware* Electrix Tweaker. Este sistema deberá funcionar en un ordenador con los sistemas operativos Windows, macOS o Linux y contar con una interfaz gráfica.

Tras el estudio realizado del estado del arte y el análisis del *hardware* se define a continuación la solución escogida.

3.3. Definición de la solución propuesta

La solución propuesta es el diseño y desarrollo de una aplicación en Java para gestionar la edición de los parámetros del controlador, en la que el usuario pueda utilizar una interfaz gráfica para seleccionar cada control, editar los valores de sus parámetros y enviarlos al Electrix Tweaker a su voluntad.

Este lenguaje de programación facilita el funcionamiento de la aplicación en los tres sistemas operativos vistos anteriormente, y además incorpora por defecto bibliotecas para gestionar las comunicaciones con el protocolo MIDI para enviar mensajes al *hardware*. Por otra parte, se cuenta con JavaFX para el diseño e implementación de la interfaz gráfica.

Control	Parámetros	Valores	Mensaje
Codificadores	Modo del anillo	Fill, Walk, Eq, Spread	MIDI
	Modo relativo	Absoluto, Relativo	MIDI
	Velocidad	1 a 7	MIDI
	Control local del anillo	On, Off	SysEx
	Mapeo del control	0 a 127	SysEx
	Canal del control	1 a 16	SysEx
	Mapeo del anillo	0 a 127	SysEx
	Canal del anillo	1 a 16	SysEx
Potenciómetros	Mapeo del control	0 a 127	SysEx
	Canal del control	1 a 16	SysEx
	Tipo de salida	CC, Nota	SysEx
Pads	Mapeo del golpe	0 a 127	SysEx
	Canal del golpe	1 a 16	SysEx
	Mapeo de la repetición	0 a 127	SysEx
	Canal del repetición	1 a 16	SysEx
	Repetición de *pads* 1 a 7	On, Off	SysEx
	Repetición de *pad* 8	On, Off	SysEx
	Umbral de Note On bajo	0 a 127	SysEx
	Umbral de Note On alto	0 a 127	SysEx
	Umbral de Note Off bajo	0 a 127	SysEx
	Umbral de Note Off alto	0 a 127	SysEx
	Cadencia de reenvío	0 a 9	SysEx
	Sensibilidad	0 a 5	SysEx
Botones	Mapeo del control	0 a 127	SysEx
	Canal del control	1 a 16	SysEx
	Tipo de salida	CC, Nota	SysEx
	Modificador de velocidad	On, Off	SysEx
	Control local del LED	On, Off	SysEx
LEDs	Estado (monocromáticas)	On, Off	MIDI
	Color (RGB)	0 a 127	MIDI

TABLA 3.2. MENSAJES MIDI QUE PERMITEN CONFIGURAR LOS PARÁMETROS DEL *HARDWARE*

Byte	Valor	Campo
0	`240`	*Byte* de apertura
1	`0`	-
2	`1`	-
3	`106`	Identificador del fabricante (Electrix)
4	`1`	Identificador del modelo (Tweaker)
5	`comando`	Número del comando a ejecutar
...	`datos`	Datos adicionales del comando
N - 1	`247`	*Byte* de cierre

TABLA 3.3. ESTRUCTURA COMÚN DE LOS MENSAJES SYSEX QUE EL TWEAKER PUEDE INTERPRETAR

4. ESPECIFICACIÓN DE REQUISITOS

Para definir de forma sistemática el comportamiento del sistema que se quiere desarrollar se usan requisitos. Un requisito es la descripción de lo que el sistema debe hacer, un servicio que ofrece o una restricción aplicada a su funcionamiento. Un conjunto de requisitos consiguen definir un sistema completo.

Para este proyecto, se definen requisitos de tipo funcional y de tipo no funcional siguiendo el siguiente formato:

ID	RF-XX
Nombre	
Prioridad	
Necesidad	
Descripción	

TABLA 4.1. FORMATO DE UN REQUISITO

La descripción de cada campo de la tabla es la siguiente:

- **ID**: Identificador único del requisito. La primera parte, compuesta por letras, define el requisito como funcional (RF) o no funcional (RNF).
- **Nombre**: Descripción breve.
- **Prioridad**: Nivel de urgencia de desarrollo del requisito. Puede ser baja, media o alta.
- **Necesidad**: Nivel de importancia del requisito para el desarrollo del sistema. Puede ser esencial, deseado u opcional.
- **Descripción**: Definición detallada.

A continuación se definen al detalle los requisitos de ambos tipos.

4.1. Requisitos funcionales

Los requisitos funcionales son aquellos que definen una función del sistema o sus componentes. Se entiende función como la especificación del comportamiento entre una entrada y una salida.

ID	RF-01
Nombre	Visualización de controles
Prioridad	Alta
Necesidad	Esencial
Descripción	La aplicación deberá mostrar todos los controles del Electrix Tweaker en la interfaz gráfica, y permitirá al usuario seleccionar con el ratón aquel para el que desee consultar o editar su valor.

TABLA 4.2. DEFINICIÓN DEL REQUISITO FUNCIONAL 1

ID	RF-02
Nombre	Visualización de los parámetros de un control
Prioridad	Alta
Necesidad	Esencial
Descripción	La interfaz gráfica de la aplicación deberá mostrar los valores de los parámetros de cada control que correspondan con el estado de la configuración actual.

TABLA 4.3. DEFINICIÓN DEL REQUISITO FUNCIONAL 2

ID	RF-03
Nombre	Edición de los parámetros
Prioridad	Alta
Necesidad	Esencial
Descripción	La aplicación permitirá al usuario editar los parámetros de la configuración actual mediante la interfaz gráfica.

TABLA 4.4. DEFINICIÓN DEL REQUISITO FUNCIONAL 3

ID	RF-04
Nombre	Restricción de valores posibles para los parámetros
Prioridad	Alta
Necesidad	Esencial
Descripción	La aplicación restringirá el rango de valores que el usuario puede introducir en los parámetros de la configuración.

TABLA 4.5. DEFINICIÓN DEL REQUISITO FUNCIONAL 4

ID	RF-05
Nombre	Estado de sincronización con el *hardware*
Prioridad	Media
Necesidad	Deseado
Descripción	La aplicación mostrará en la interfaz gráfica si la configuración actual está sincronizada con el *hardware* o si se ha hecho algún cambio desde el último volcado.

TABLA 4.6. DEFINICIÓN DEL REQUISITO FUNCIONAL 5

ID	RF-06
Nombre	Volcado de configuración con el *hardware*
Prioridad	Alta
Necesidad	Esencial
Descripción	La aplicación enviará la configuración actual al *hardware*, modificando los parámetros de los controles y marcando el estado de sincronización como cumplido.

TABLA 4.7. DEFINICIÓN DEL REQUISITO FUNCIONAL 6

ID	RF-07
Nombre	Reconexión con el *hardware*
Prioridad	Media
Necesidad	Deseado
Descripción	Si el usuario intenta hacer un volcado, la aplicación comprobará si el *hardware* está conectado, y en caso negativo intentará realizar una reconexión.

TABLA 4.8. DEFINICIÓN DEL REQUISITO FUNCIONAL 7

ID	RF-08
Nombre	Manejo de fallo de la reconexión
Prioridad	Media
Necesidad	Deseado
Descripción	Si la aplicación intenta realizar una reconexión y la operación falla, el volcado no se realizará.

TABLA 4.9. DEFINICIÓN DEL REQUISITO FUNCIONAL 8

ID	RF-09
Nombre	Establecimiento de conexión inicial
Prioridad	Alta
Necesidad	Esencial
Descripción	Al iniciar la aplicación, ésta intentará realizar una conexión con el *hardware*.

TABLA 4.10. DEFINICIÓN DEL REQUISITO FUNCIONAL 9

ID	RF-10
Nombre	Guardado de configuración a fichero
Prioridad	Media
Necesidad	Opcional
Descripción	La aplicación permitirá almacenar una configuración actual en un fichero.

TABLA 4.11. DEFINICIÓN DEL REQUISITO FUNCIONAL 10

ID	RF-11
Nombre	Carga de configuración desde fichero
Prioridad	Media
Necesidad	Opcional
Descripción	La aplicación permitirá cargar una configuración actual desde un fichero.

TABLA 4.12. DEFINICIÓN DEL REQUISITO FUNCIONAL 11

ID	RF-12
Nombre	Propagación de cambios
Prioridad	Baja
Necesidad	Opcional
Descripción	La aplicación permitirá al usuario copiar el valor de los parámetros de un control a otros del mismo tipo.

TABLA 4.13. DEFINICIÓN DEL REQUISITO FUNCIONAL 12

ID	RF-13
Nombre	Mantenimiento de mapeos en propagación
Prioridad	Baja
Necesidad	Opcional
Descripción	La aplicación permitirá que los campos de mapeos de nota, CC y canal MIDI no se copien con las propagaciones.

TABLA 4.14. DEFINICIÓN DEL REQUISITO FUNCIONAL 13

ID	RF-14
Nombre	Propagación en fila
Prioridad	Baja
Necesidad	Opcional
Descripción	La aplicación permitirá realizar una propagación a todos los controles de la columna a la que pertenezca uno de ellos si todos los del mismo tipo están dispuestos en cuadrícula.

TABLA 4.15. DEFINICIÓN DEL REQUISITO FUNCIONAL 14

ID	RF-15
Nombre	Propagación en columna
Prioridad	Baja
Necesidad	Opcional
Descripción	La aplicación permitirá realizar una propagación a todos los controles de la fila a la que pertenezca unó de ellos si todos los del mismo tipo están dispuestos en cuadrícula.

TABLA 4.16. DEFINICIÓN DEL REQUISITO FUNCIONAL 15

4.2. Requisitos no funcionales

Los requisitos no funcionales son aquellos que especifican un criterio a seguir para evaluar el funcionamiento del sistema, en lugar de comportamientos específicos. En este caso, se centran en restringir el diseño de la arquitectura.

ID	RNF-01
Nombre	Conexión con el Electrix Tweaker
Prioridad	Alta
Necesidad	Esencial
Descripción	El sistema podrá conectarse y enviar datos usando el protocolo MIDI al *hardware* Electrix Tweaker a través de una conexión USB.

TABLA 4.17. DEFINICIÓN DEL REQUISITO NO FUNCIONAL 1

ID	RNF-02
Nombre	Condición de multiplataforma
Prioridad	Alta
Necesidad	Esencial
Descripción	La aplicación deberá poder ejecutarse en las versiones más recientes de los sistemas operativos Windows, macOS y Linux sin que los cambios necesarios para el despliegue afecten al código fuente.

TABLA 4.18. DEFINICIÓN DEL REQUISITO NO FUNCIONAL 2

ID	RNF-03
Nombre	Uso de JavaFX
Prioridad	Alta
Necesidad	Esencial
Descripción	Se usará JavaFX para construir la interfaz gráfica de la aplicación.

TABLA 4.19. DEFINICIÓN DEL REQUISITO NO FUNCIONAL 3

ID	RNF-04
Nombre	Diseño de la interfaz gráfica
Prioridad	Alta
Necesidad	Esencial
Descripción	La interfaz gráfica de la aplicación deberá mostrar una única ventana durante toda su ejecución, exceptuando mensajes de alertas y errores.

TABLA 4.20. DEFINICIÓN DEL REQUISITO NO FUNCIONAL 4

ID	RNF-05
Nombre	Notificación de alertas y errores
Prioridad	Media
Necesidad	Deseado
Descripción	La aplicación mostrará al usuario los errores y avisos que se produzcan en la ejecución, informando sobre las acciones que se llevarán o no a cabo inmediatamente después.

TABLA 4.21. DEFINICIÓN DEL REQUISITO NO FUNCIONAL 5

ID	RNF-06
Nombre	Formato de ficheros de guardado local
Prioridad	Alta
Necesidad	Deseado
Descripción	Los ficheros donde la aplicación almacene la configuración tendrán el formato JSON.

TABLA 4.22. DEFINICIÓN DEL REQUISITO NO FUNCIONAL 6

5. DISEÑO DE CASOS DE USO

Con los requisitos especificados, podemos refinar el diseño del sistema definiendo casos de uso. Los casos de uso representan listas de acciones que los actores del sistema pueden realizar, y permiten agrupar requisitos comunes definiendo lo que los actores pueden hacer con el sistema final.

En primer lugar se identifican los casos de uso y los actores del sistema y se representa de forma gráfica. Después, se realiza una descripción detallada de cada caso de uso.

5.1. Diagrama de casos de uso

En el caso de este proyecto, el diagrama resulta simple al existir un único actor, el usuario de la aplicación. Además, todos los casos de uso quedan contenidos dentro del sistema de la aplicación editor, como se puede observar en la figura 5.1:

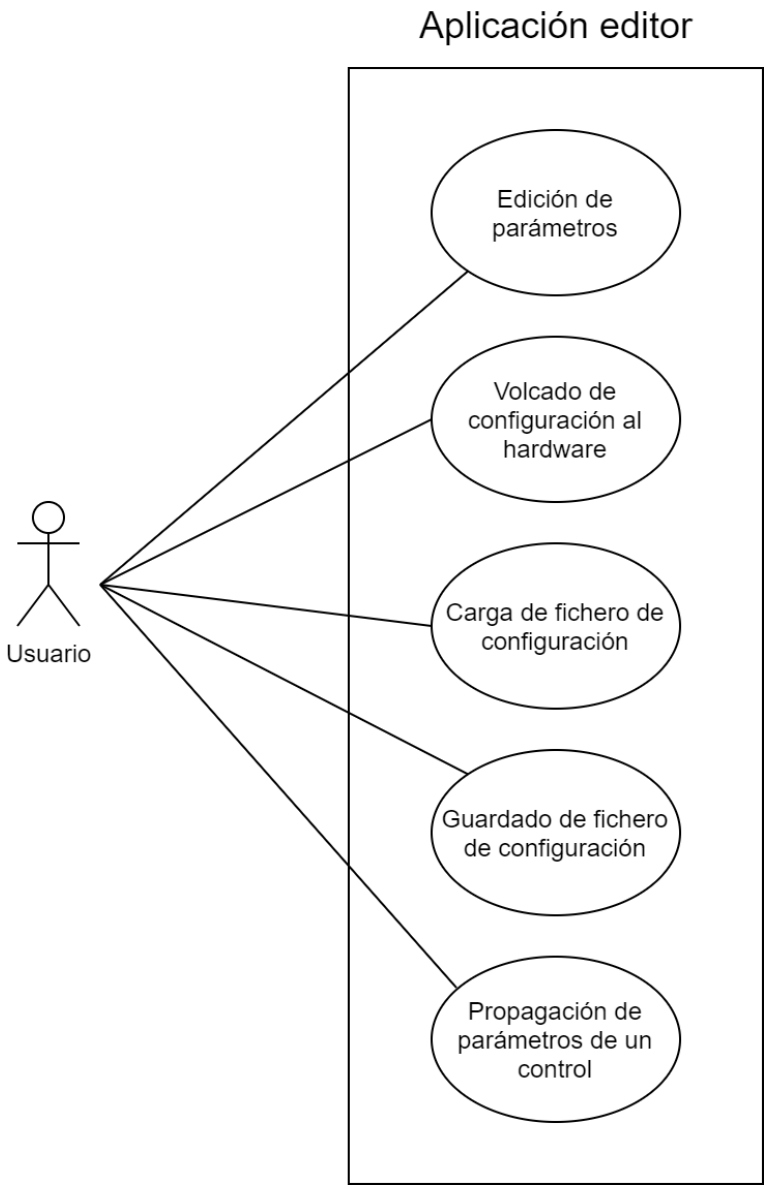


Fig. 5.1. Diagrama de casos de uso del sistema

5.2. Definición de casos de uso

En esta sección se describen los casos de uso identificados en el diagrama 5.1, utilizando el siguiente formato de tabla:

ID	CU-XX
Nombre	
Actor	
Objetivos	
Precondiciones	
Postcondiciones	
Escenario básico	
Escenario alternativo	
Requisitos	

TABLA 5.1. FORMATO DEL CASO DE USO

La definición de cada campo de la tabla anterior es:

- **ID**: Identificador único del caso de uso. La primera parte es siempre CU (Caso de Uso), seguido del descriptor numérico separado por un guión.
- **Nombre**: Descripción de la tarea que el actor lleva a cabo.
- **Actor**: Agente (persona o sistema) que inicia este caso de uso o que participa en él.
- **Objetivos**: Cambios en el sistema que el actor quiere conseguir, propósito del caso de uso.
- **Precondiciones**: Condiciones que se deben cumplir para que se dé un progreso correcto del caso de uso.
- **Postcondiciones**: Condiciones que debe presentar el sistema tras la finalización de las acciones del caso de uso.
- **Escenario básico**: Progresión de eventos que se realizarán en el caso de uso.
- **Escenario alternativo**: Progresión de eventos alternativa que se pueden realizar en el caso de uso.
- **Requisitos**: Requisitos funcionales relacionados con el caso de uso, especificados en la sección 4.1.

Los casos de uso identificados en el sistema son los siguientes:

ID	CU-01
Nombre	Edición de parámetros
Actor	Usuario
Objetivos	Visualización y establecimiento del valor de un parámetro de la configuración actual.
Precondiciones	La aplicación está en ejecución y la comprobación de la conexión ha sido completada (conectado, o no conectado y diálogo de aviso cerrado).
Postcondiciones	La ventana muestra el nuevo valor.
Escenario básico	1. El usuario selecciona un control en el panel derecho de la aplicación. 2. La aplicación muestra los parámetros del control seleccionado en el panel izquierdo. 3. El usuario utiliza los campos de entrada de ese panel para modificar uno o varios parámetros. 4. La aplicación modifica la configuración actual con esos nuevos valores.
Requisitos	RF-01, RF-02, RF-03, RF-04

TABLA 5.2. DEFINICIÓN DEL CASO DE USO 1

ID	CU-02
Nombre	Volcado de configuración al *hardware*
Actor	Usuario
Objetivos	Modificación de los parámetros de configuración del Electrix Tweaker de acuerdo con la configuración actual.
Precondiciones	La aplicación está en ejecución y la comprobación de la conexión ha sido completada (conectado, o no conectado y diálogo de aviso cerrado).
Postcondiciones	Los parámetros del *hardware* reflejan los de la configuración actual y el estado de sincronización pasa a ser "sincronizado".
Escenario básico	1. El usuario pulsa el botón de volcado en el panel derecho de la aplicación. 2. Los parámetros del *hardware* se actualizan para coincidir con los de la configuración actual. 3. La aplicación cambia el estado de sincronización a "sincronizado".
Escenario alternativo	1. El usuario pulsa el botón de volcado en el panel derecho de la aplicación con el *hardware* desconectado. 2. Aparece una ventana de aviso pidiendo al usuario de la conexión del *hardware*. 3. El usuario conecta el *hardware*. 4. El usuario pulsa OK en la ventana de aviso. 5. La aplicación realiza la conexión con el *hardware*. 6. Los parámetros del *hardware* se actualizan para coincidir con los de la configuración actual.
Requisitos	RF-05, RF-06, RF-07, RF-08, RF-09

TABLA 5.3. DEFINICIÓN DEL CASO DE USO 2

ID	CU-03
Nombre	Carga de fichero de configuración
Actor	Usuario
Objetivos	Lectura de un fichero local y modificación de la configuracion actual de acuerdo con su contenido.
Precondiciones	La aplicación está en ejecución y la comprobación de la conexión ha sido completada (conectado, o no conectado y diálogo de aviso cerrado).
Postcondiciones	Los parámetros de la configuración actual tienen el valor indicado en el fichero.
Escenario básico	1. El usuario pulsa el botón de cargado desde fichero situado en el panel derecho de la aplicación. 2. Aparece la ventana de selección de fichero. 3. El usuario introduce la ruta y el nombre del fichero a leer. 4. El usuario pulsa en el botón de cargado. 5. La ventana de selección de fichero se cierra. 6. La aplicación lee el fichero y modifica su configuración actual acorde a su contenido.
Requisitos	RF-11

TABLA 5.4. DEFINICIÓN DEL CASO DE USO 3

ID	CU-04
Nombre	Guardado de fichero de configuración
Actor	Usuario
Objetivos	Escritura de la configuración actual en un fichero local.
Precondiciones	La aplicación está en ejecución y la comprobación de la conexión ha sido completada (conectado, o no conectado y diálogo de aviso cerrado).
Postcondiciones	Los parámetros del fichero tienen el valor indicado en la configuración actual.
Escenario básico	1. El usuario pulsa el botón de guardado en fichero situado en el panel derecho de la aplicación. 2. Aparece la ventana de selección de fichero. 3. El usuario introduce la ruta y el nombre del fichero a escribir. 4. El usuario pulsa en el botón de guardado. 5. La ventana de selección de fichero se cierra. 6. La aplicación escribe en el fichero la configuración actual.
Requisitos	RF-10

TABLA 5.5. DEFINICIÓN DEL CASO DE USO 4

ID	CU-05
Nombre	Propagación de parámetros de un control
Actor	Usuario
Objetivos	Copia de los valores de los parámetros de un control a los demás del mismo tipo.
Precondiciones	La aplicación está en ejecución y la comprobación de la conexión ha sido completada (conectado, o no conectado y diálogo de aviso cerrado).
Postcondiciones	Los parámetros de los controles de la misma categoría.
Escenario básico	1. El usuario pulsa el botón de propagación a todos los controles de la misma categoría. 2. La aplicación copia los valores de los parámetros del control seleccionado actualmente a los demás del mismo tipo.
Escenario alternativo	1. El usuario marca la casilla de mantener los mapeos MIDI en las propagaciones. 2. El usuario pulsa el botón de propagación a todos los controles de la misma categoría. 3. La aplicación copia los valores de los parámetros del control seleccionado actualmente a los demás del mismo tipo, exceptuando los mapeos MIDI.
Requisitos	RF-12, RF-13, RF-14, RF-15

TABLA 5.6. DEFINICIÓN DEL CASO DE USO 5

5.3. Matriz de trazabilidad

La siguiente tabla relaciona los requisitos especificados en la sección 4 con los casos de uso definidos previamente. Esto permite comprobar la completitud de los casos de uso, ya que cada requisito queda reflejado en, al menos, uno de ellos.

	CU-01	CU-02	CU-03	CU-04	CU-05
RF-01	O				
RF-02	O				
RF-03	O				
RF-04	O				
RF-05		O			
RF-06		O			
RF-07		O			
RF-08		O			
RF-09		O			
RF-10				O	
RF-11			O		
RF-12					O
RF-13					O
RF-14					O
RF-15					O

TABLA 5.7. MATRIZ DE TRAZABILIDAD DE CASOS DE USO Y REQUISITOS

6. ARQUITECTURA DEL SISTEMA

Para la descripción de la arquitectura de la solución propuesta se usan dos tipos de vistas diferentes del modelo 4+1 [32]:

- **Vista lógica**: Describe la funcionalidad que el sistema ofrece al usuario.
- **Vista de desarrollo**: Describe el sistema desde el punto de vista del programador.

El resto de vistas se consideran de menor relevancia para este proyecto y han sido omitidas.

6.1. Vista lógica

En esta sección se describe la vista lógica del sistema siguiendo el paradigma de programación orientada a objetos. Se trata de estructurar el *software* en objetos, compuestos por atributos (datos contenidos en el objeto) y métodos (funciones que realizan ciertas acciones sobre el objeto al que pertenecen) [33]. Las clases determinan la estructura genérica de los objetos, que toman valores concretos al ser instanciadas.

Para representar el diseño de las clases y sus relaciones en un sistema se utilizan los diagramas de clases. En el caso de este proyecto, se muestra el diseño de clases realizado en la figura 6.1.

A continuación se describen las clases indicadas en el diagrama de clases y su función dentro del sistema, así como sus métodos públicos (excepto los constructores):

- `MonoLed`: Representación de un LED monocromático (rojo o azul) del *hardware*, usados en los botones de los potenciómetros rotativos, *pads* y botones de navegación.
  - `channel`: Canal MIDI por el que debe recibir mensajes para cambiar de estado.
  - `mapping`: CC que debe recibir para cambiar de estado.

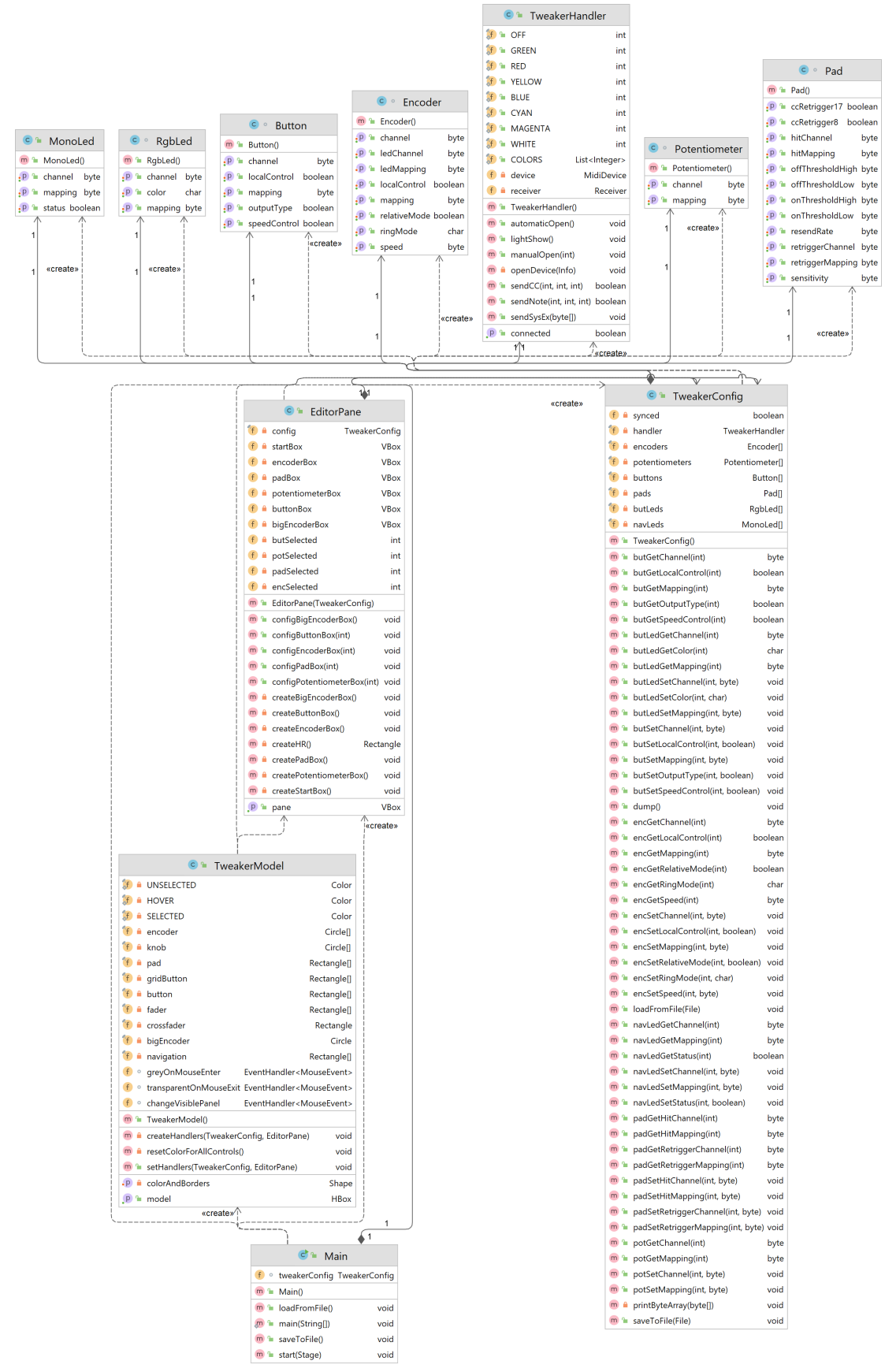


Fig. 6.1. Diagrama de clases

- `status`: Estado de encendido (iluminado o apagado).

- **`RgbLed`**: Representación de un LED multicolor (RGB), usados para los botones de la cuadrícula y otros botones auxiliares.
  - `channel`: Canal MIDI por el que debe recibir mensajes para cambiar de estado.
  - `color`: Tipo de iluminación (apagado, verde, rojo, amarillo, azul, cian, magenta o blanco).
  - `mapping`: CC que debe recibir para cambiar de estado.

- **`Button`**: Representación de un botón del *hardware*.
  - `channel`: Canal MIDI por el que se envían los mensajes cuando el control cambia de estado.
  - `localControl`: Vincula o desvincula la iluminación de la luz LED asignada al botón con su estado.
  - `mapping`: Nota o CC que envía al cambiar de estado (pulsar y soltar).
  - **outputType**: Tipo de mensaje MIDI, nota o CC, que envía el botón.
  - **speedControl**: Activa el botón como modificador de velocidad de los codificadores rotativos.

- **`Encoder`**: Representación de un codificador rotativo junto con su correspondiente anillo LED.
  - `channel`: Canal MIDI por el que se envían los mensajes cuando el control cambia de estado.
  - `ledChannel`: Canal MIDI por el que debe recibir mensajes para cambiar la iluminación del anillo LED.
  - `ledMapping`: CC que debe recibir para cambiar la iluminación del anillo LED.
  - `localControl`: Permite vincular o desvincular la iluminación del anillo LED de la posición del codificador.
  - `mapping`: CC que envía al cambiar de estado (rotar).

- `relativeMode`: Activa o desactiva el modo relativo.
- `ringMode`: Cambia entre los distintos modos de iluminación del anillo LED (rellenar, caminar, EQ, o distribuir).
- `speed`: Velocidad de avance del codificador en modo absoluto (paso).

- **`Potentiometer`**: Representación de un potenciómetro lineal o rotativo.
  - `channel`: Canal MIDI por el que se envían los mensajes cuando el control cambia de estado.
  - `mapping`: CC que envía al cambiar de estado (desplazarse o rotar).

- **`Pad`**: Representación de un *pad* sensible a la presión.
  - `ccRetrigger17`: Activa o desactiva la función de disparo repetido para todos los *pads* excepto el 8.
  - `ccRetrigger8`: Activa o desactiva la función de disparo repetido únicamente para el *pad* 8.
  - `hitChannel`: Canal MIDI por el que se envían los mensajes en el primer golpe.
  - `hitMapping`: Nota que se envía en el primer golpe.
  - `offThresholdHigh`: Umbral alto de presión de apagado.
  - `offThresholdLow`: Umbral bajo de presión de apagado.
  - `onThresholdHigh`: Umbral alto de presión de encendido.
  - `onThresholdLow`: Umbral bajo de presión de encendido.
  - `resendRate`: Cadencia de envío de mensajes durante el disparo repetido.
  - `retriggerChannel`: Canal MIDI por el que se envían los mensajes durante el disparo repetido.
  - `retriggerMapping`: CC que se envía durante el disparo repetido.
  - `sensitivity`: Sensibilidad del *pad* a la fuerza transmitida por el usuario.

- **`TweakerHandler`**: Establece la comunicación entre el sistema y el Electrix Tweaker. Pensado como *singleton*, se instancia en la clase *TweakerConfig*.

- `automaticOpen()`: Busca un Electrix Tweaker conectado al ordenador y, si lo encuentra, lo conecta al sistema.
- `lightShow()`: Función de demostración que realiza llamadas a `automaticOpen()` y `sendNote()` para probar su funcionamiento durante el desarrollo.
- `manualOpen(int)`: Conecta el dispositivo MIDI indicado por argumentos.
- `sendCC(int, int, int)`: Envía un mensaje CC dados un canal, un número de CC y un valor al dispositivo conectado en ese momento.
- `sendNote(int, int, int)`: Envía una nota dados un canal, un número de nota y una velocidad al dispositivo conectado en ese momento.
- `sendSysEx(byte[])`: Envía una cadena de *bytes* como mensaje SysEx al dispositivo conectado en ese momento.

- **`TweakerConfig`**: Representa una configuración del Electrix Tweaker. Pensado también como *singleton*, contiene una instancia de *TweakerHandler* para las funciones de comunicación con el *hardware*.
  - `xxxGetYyy(int)`: Conjunto de funciones que devuelven el valor del parámetro Yyy del control de tipo xxx pasado por argumentos.
  - `xxxSetYyy(int)`: Conjunto de funciones que modifican el valor del parámetro Yyy del control de tipo xxx pasado por argumentos.
  - `dump()`: Envía la configuración actual, tal y como está representada en la instancia de la clase, al *hardware*.
  - `loadFromFile(File)`: Carga la configuración desde un archivo, sobreescribiendo los valores de los campos de los controles.
  - `saveToFile(File)`: Guarda los valores de los campos de los controles a un archivo, cargable con la función `loadFromFile()`.
- **`EditorPane`**: Contiene los paneles de edición de parámetros de la interfaz gráfica, y se encarga de sincronizarlos con la configuración.
  - `pane`: Contiene todos los paneles de cada tipo de control.

    - `configXxxBox(int)`: Para cada control indicado por argumento de tipo Xxx, carga los valores de los parámetros de ese control y los plasma en el panel de la interfaz.

- **`TweakerModel`**: Panel del modelo gráfico del Electrix Tweaker, que gestiona la selección de cada control por parte del usuario mostrando y configurando los paneles de la clase `EditorPane`.

    - `setHandlers(TweakerConfig, EditorPane)`: Enlaza los paneles de edición de parámetros de la interfaz gráfica con la configuración del *hardware*.
    - `model`: Proporciona acceso al modelo gráfico del Tweaker.

- **`Main`**: Clase principal del programa, que instancia todos los objetos necesarios a nivel de interfaz gráfica y proporciona las ventanas de carga y guardado de ficheros de configuración.

    - `loadFromFile()`: Muestra la ventana de selección de fichero y llama a la función `loadFromFile(File)` de la clase `TweakerConfig` usando el fichero abierto como argumento.
    - `main(String[])`: Inicio del programa.
    - `saveToFile()`: Muestra la ventana de creación o selección de fichero y llama a la función `saveToFile(File)` de la clase `TweakerConfig` usando el fichero abierto como argumento.
    - `start(Stage)`: Inicializa la interfaz gráfica de JavaFX.

6.2. Vista de desarrollo

En esta sección se analizan los componentes del sistema siguiendo el patrón Modelo-Vista-Controlador (MVC) [34]. Se trata de un patrón que divide el sistema en:

- **Modelo**: Contiene todos los datos de la aplicación.
- **Vista**: Se encarga de mostrar los datos al usuario.
- **Controlador**: Modifica y gestiona los datos y su visualización.

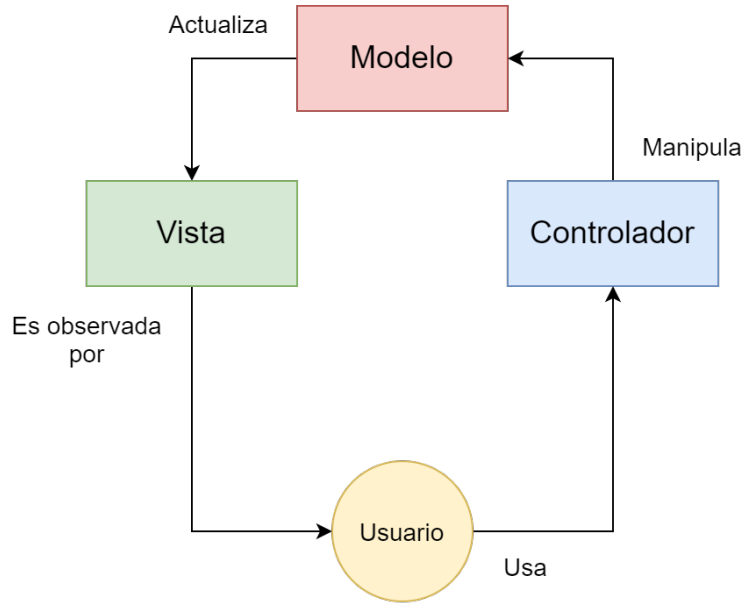


Fig. 6.2. Diagrama de interacciones en el patrón MVC

Estos tres componentes interaccionan entre ellos como se indica en la figura 6.2.

En el caso de este proyecto y siguiendo la distinción de las clases hechas en el apartado anterior, podemos identificar cuáles corresponden a cada entidad del patrón MVC:

- **Modelo**: `TweakerConfig`, clase donde se almacena la configuración actual del *hardware*.
- **Vista**: `EditorPane`, clase que muestra los datos del modelo en la interfaz gráfica.
- **Controlador**: `TweakerModel`, clase que intercambia las vistas modificando los datos que muestran acorde al modelo.

7. IMPLEMENTACIÓN Y DISTRIBUCIÓN

A continuación se incluyen detalles de implementación del proyecto en relación con el desarrollo de la aplicación a nivel técnico, así como el análisis y diseño de la interfaz gráfica y experiencia de usuario. Finalmente, se describen el formato de fichero local de configuración y el sistema de distribución empleado para la aplicación.

7.1. Entorno de desarrollo

El entorno de desarrollo (programación) de un proyecto es el conjunto de herramientas tanto *software* como *hardware* que facilitan la fase de programación del mismo, en tareas como edición de código y compilación [35].

En general, la selección de herramientas del entorno se realiza en base a las tecnologías que vayan a ser usadas en el proyecto en cuestión. En este caso, el factor limitante es la elección de Java como lenguaje de programación para el desarrollo del código. En base a esto, el entorno de desarrollo escogido es:

1. **Ordenador personal** con un procesador Intel i5-10400F y 16 GB de memoria RAM, con al menos un puerto USB compatible con el estándar USB 2.0 para permitir la conexión con el Electrix Tweaker.

2. Sistema operativo **Windows 10 Educación**. Escogido por facilidad de uso y por conveniencia, pero al ser un desarrollo multiplataforma, no es especialmente relevante para el entorno.

3. **OpenJDK 15**, en su versión 15.0.2, la más actualizada y estable en la fecha de inicio del proyecto. Contiene el compilador y máquina virtual de Java, entre otras utilidades de desarrollo. No es posible desarrollar una aplicación en Java sin un JDK (*Java Development Kit*) instalado.

4. **JavaFX SDK 15**, en su versión 15.0.1, la más actualizada y estable en la fecha de inicio del proyecto. Se trata de la biblioteca para generar las interfaces gráficas usadas en el proyecto.

5. **Gson**, en su versión 2.8.7, la más actualizada y estable en la fecha de inicio del proyecto. Es la biblioteca usada en la aplicación para la lectura y escritura de ficheros con formato JSON.

6. **IntelliJ IDEA 2020.3 Community Edition**, la versión más actualizada y estable en la fecha de inicio del proyecto. Es el Entorno de Desarrollo Integrado (IDE) usado para gestionar los ficheros fuente del proyecto, así como el sistema de control de versiones.

7. **Git** como Sistema de Control de Versiones (VCS).

8. **GitHub** como plataforma de almacenamiento de repositorios de Git, con un repositorio privado para el desarrollo de la aplicación.

9. **WiX Toolset** 3.11.2 para generar los instaladores de la aplicación en Windows.

La elección de herramientas y versiones se ha mantenido constante durante el desarrollo (a excepción del sistema operativo, que se ha visto actualizado varias veces desde el inicio del proyecto). Esto se ha hecho para evitar posibles problemas de integración, como:

- Abandono de funcionalidad requerida para el proyecto en mitad de la ejecución del proyecto por parte del desarrollador de alguna herramienta.
- Modificación de variables, clases o métodos en una biblioteca en uso, que requieran cambios exhaustivos en el código de la aplicación.
- Abandono del soporte para alguna de las plataformas (Windows, macOS o Linux).

Se requerirá trabajo futuro para actualizar a las últimas versiones de todas las herramientas de desarrollo y aprovechar las mejoras en seguridad y rendimiento, intentando minimizar los cambios requeridos en el código y el abandono de funcionalidad.

7.2. Diseño de la interfaz gráfica

Un factor importante del desarrollo de este proyecto es el diseño de una interfaz gráfica que sea fácil y rápida de usar. En este caso, este diseño está guiado por:

- La interfaz gráfica existente de la aplicación que queremos sustituir. Se pueden escoger las características deseables para el usuario y mejorarlas, además de añadir otras nuevas.
- Otras aplicaciones que sirven el mismo propósito, como Ctrlr (vista en la sección 2.4.2) o MF Utility (el editor oficial del controlador MIDI Fighter Twister).
- Reglas del diseño ético [36], como:
  - Apuntar a conseguir la usabilidad máxima, evitando confundir al usuario previniendo consecuencias no deseadas, a través de la simplicidad y la claridad.
  - Mantener la privacidad del usuario a salvo en la mayor medida posible.
  - Evitar el diseño que genera frustración en la experiencia de usuario específicamente para intentar vender otra versión del producto.
- Las heurísticas de usabilidad de Nielsen, que definen 10 principios generales para el diseño de interacción humano-máquina [37].

7.2.1. Análisis

El primer paso es analizar tanto la interfaz del editor original del Tweaker como la del editor del Fighter Twister e identificar características de diseño deseables.

En la interfaz del editor original, mostrada en la figura 7.1, se puede observar que:

- Consta de una única ventana.
- No hace uso de pestañas o desplegables que muestren partes del editor y oculten otras que sean necesarias al mismo tiempo.
- Muestra un esquema del controlador y permite interactuar con él.
- Tiene una distribución clara, separando los parámetros editables y los botones en secciones.
- Utiliza los colores para resaltar información importante, como el control que está actualmente seleccionado, de forma más funcional que estilística.

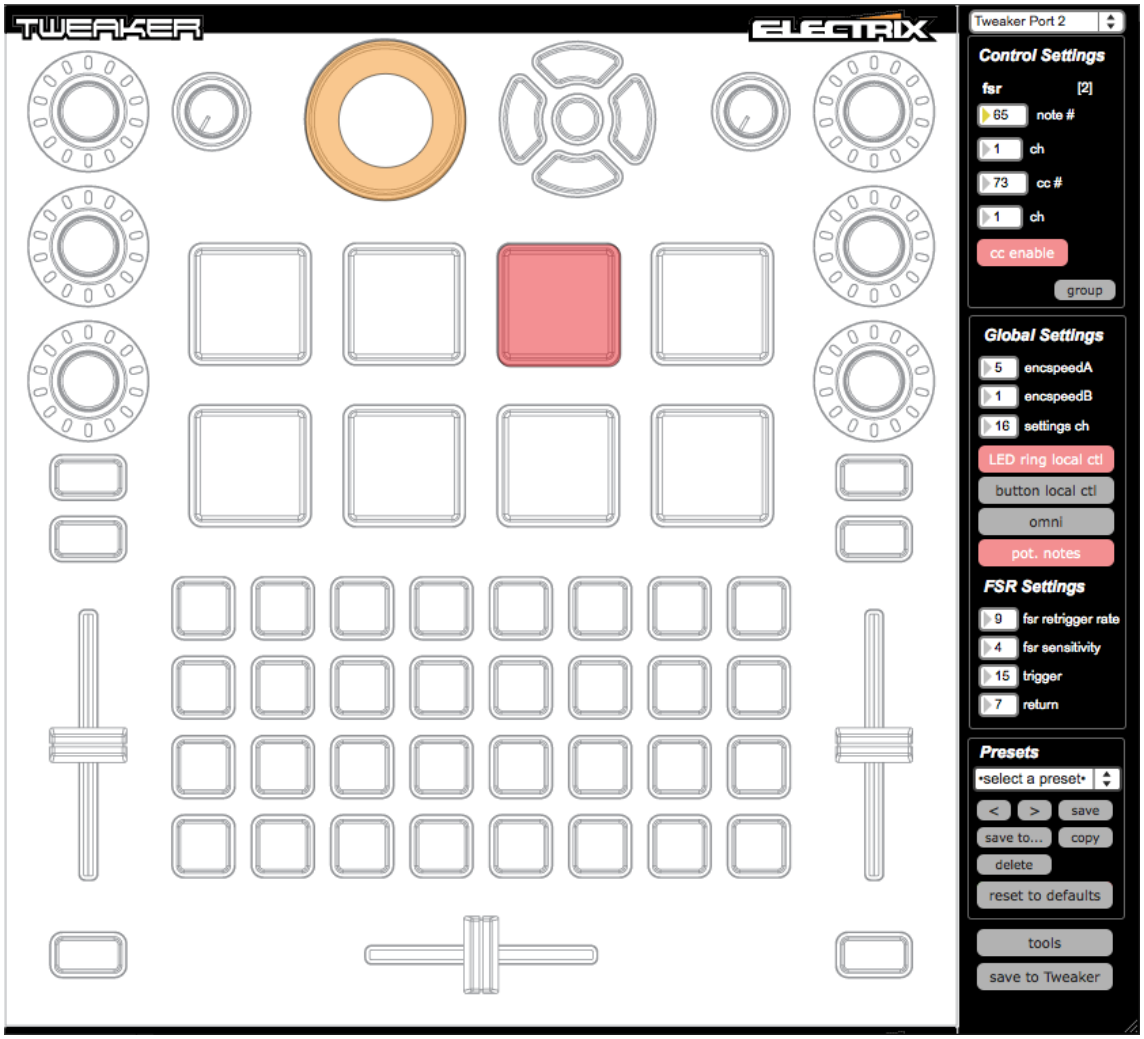


Fig. 7.1. Interfaz del editor original

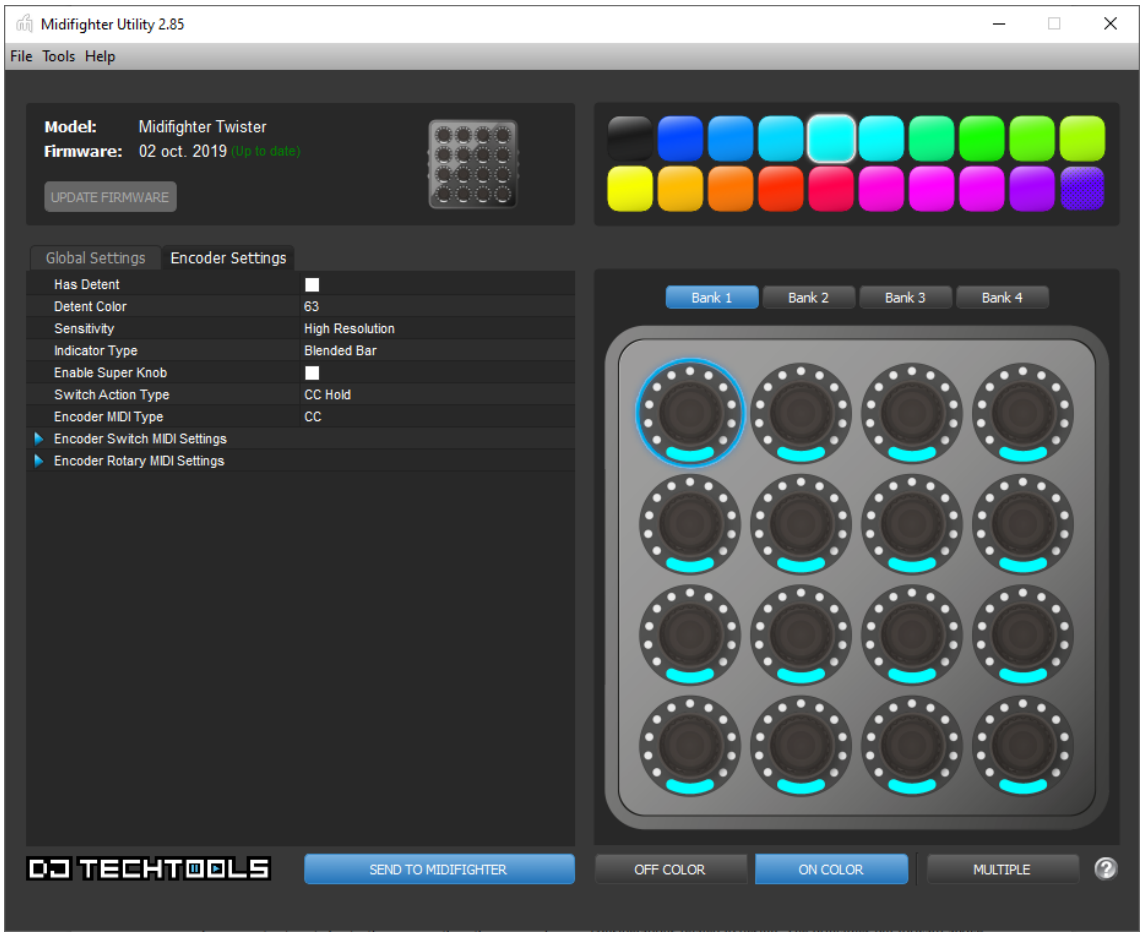


Fig. 7.2. Interfaz del editor del Fighter Twister

El mayor aspecto negativo a nivel de usabilidad viene dado por el tamaño de la ventana principal y los parámetros, que tienen dimensiones muy reducidas con respecto al gráfico de la disposición de los controles y resultan difíciles de leer y editar.

El caso del editor del Fighter Twister, mostrado en la figura 7.2, es muy similar:

- Consta de una única ventana.
- No hace uso de pestañas o desplegables.
- Muestra un esquema del controlador y permite interactuar con él.
- Tiene una distribución clara, separando los parámetros editables y los botones en secciones.

En este caso, el sombreado de color que resalta el control seleccionado es menos visible, pero la disposición y tamaño de los parámetros y la interfaz en general es más adecuada.

7.2.2. Propuesta de diseño

Siguiendo las reglas del diseño ético descritas anteriormente y las observaciones hechas de las dos interfaces existentes, se realizan bocetos con el *software* de edición gráfica Inkscape para planificar el diseño de la interfaz gráfica de este proyecto antes de su implementación. El primer boceto se muestra en la figura 7.3, donde se puede ver que:

- Se diseña la aplicación en una única ventana.
- Las dimensiones de la interfaz se incrementan en horizontal para aprovechar el formato de los monitores 16:9, 16:10 y 21:9.
- Se divide la información de la ventana en dos partes: A la izquierda, los parámetros del control seleccionado. A la derecha, la representación gráfica del *hardware* y los controles globales de la aplicación. El panel izquierdo variará dependiendo de la selección del usuario, mientras que el derecho será constante.
- Se incluye un gráfico que muestra una representación realista de la disposición de los controles, pero con la que no se puede interactuar.
- Se utiliza el color exclusivamente para resaltar los controles seleccionados.
- Se hace uso de pestañas para mostrar unos controles u otros.
- La cantidad de parámetros que se muestran por cada control es muy limitada y no permite incluir más.

En la segunda iteración del boceto, mostrado en la figura 7.4, se mejoran ciertos aspectos del diseño:

- Se eliminan las pestañas, haciendo la representación del *hardware* interactuable.
- En el panel izquierdo se muestra exclusivamente la información del control seleccionado por el usuario en la representación gráfica del controlador, dejando espacio para un mayor número de parámetros y botones adicionales, incluyendo la propagación.

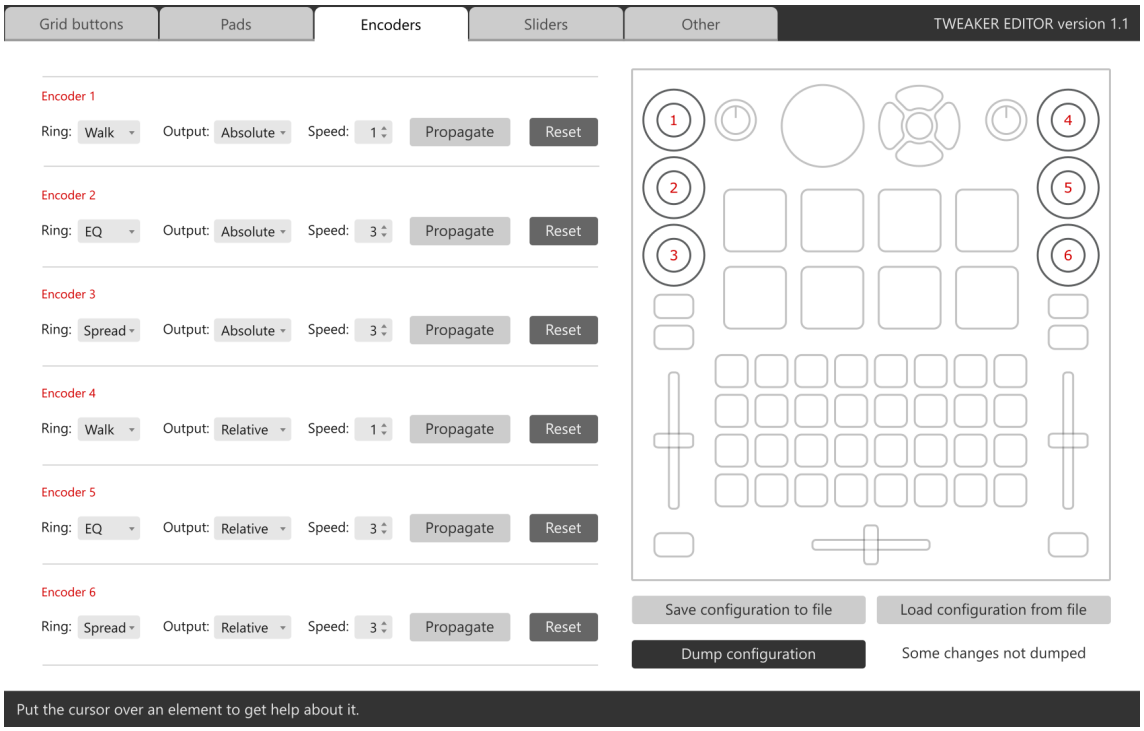


Fig. 7.3. Primer boceto de la interfaz gráfica

- La distribución pasa a ser más clara y accesible, y permite añadir más parámetros y funciones asociadas a cada control, como el reinicio de valores (“Reset”) y botones de propagación avanzada.
- Se elimina temporalmente el color.

El segundo boceto se usa como guía para la implementación de la interfaz gráfica y se consigue la primera versión, que se muestra en la figura 7.5. La disposición es muy similar a la del boceto final, aunque se eliminan las barras de título y de estado (principio y final verticales de la interfaz, respectivamente).

Este diseño se amplía y refina en la segunda iteración de la implementación, que se muestra en la figura 7.6, donde se puede observar que:

- La interfaz cuenta con el diseño dividido en dos paneles, donde:
    1. El panel derecho muestra controles globales y la representación gráfica del controlador con la que el usuario puede interaccionar para modificar el panel izquierdo.
    2. El panel izquierdo muestra los parámetros editables del control seleccionado

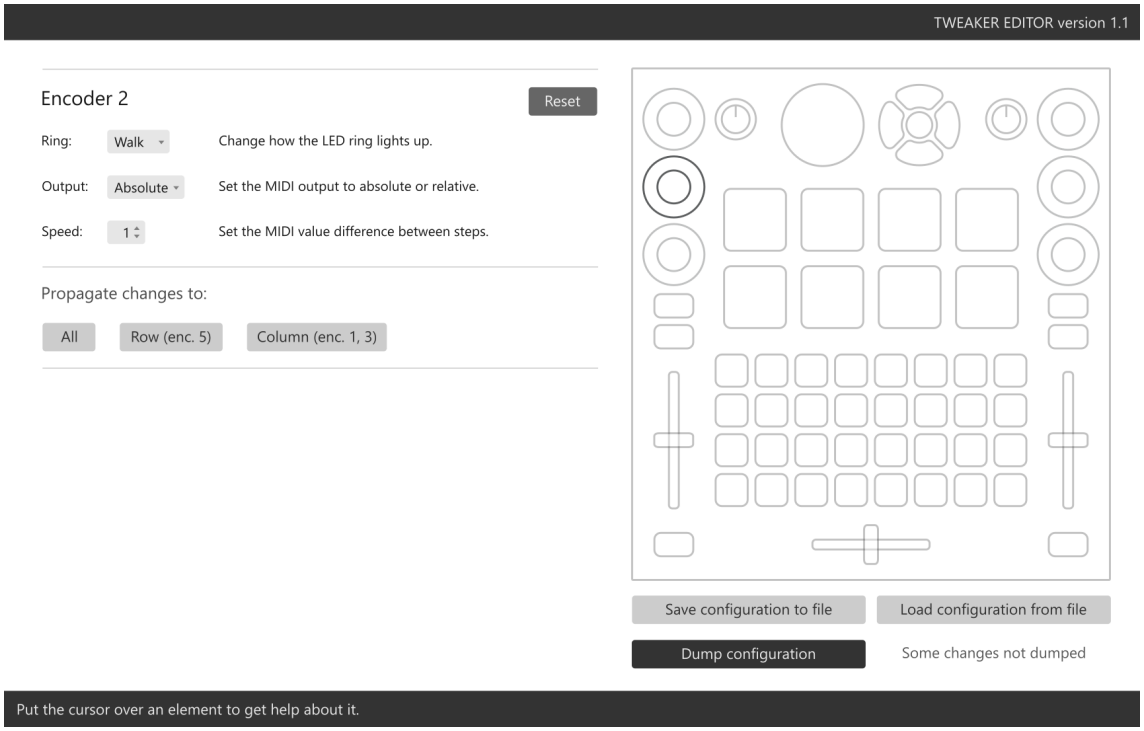


Fig. 7.4. Segundo boceto de la interfaz gráfica

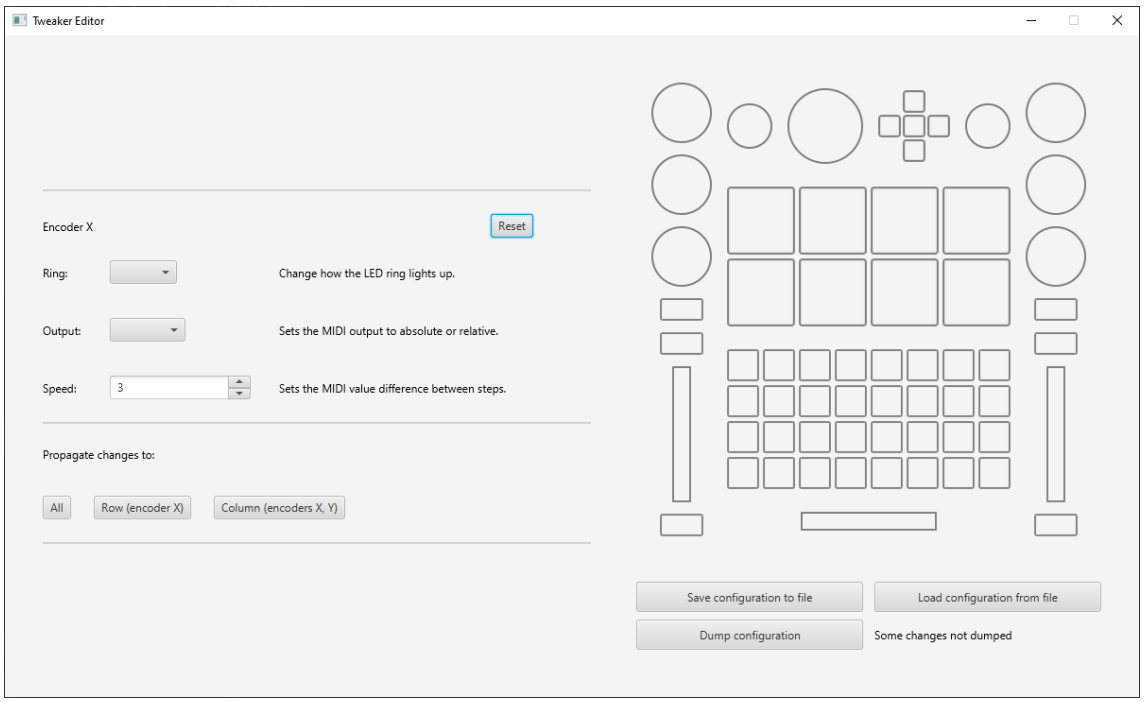


Fig. 7.5. Primera versión de la interfaz implementada con JavaFX

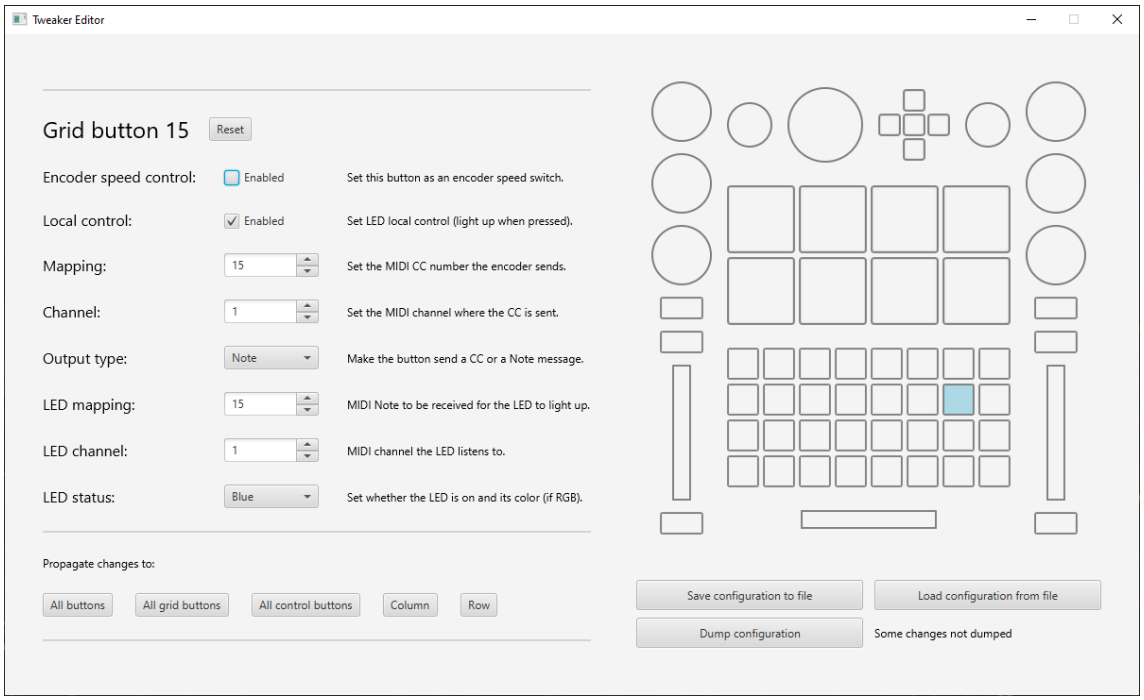


Fig. 7.6. Segunda versión de la interfaz implementada con JavaFX

por el usuario, además de controles adicionales como la propagación.

- Se indica claramente el control que está seleccionado, tanto en el gráfico con un resaltado de color azul como en el panel de edición con un título que indica el nombre del control y su número.
- Los parámetros tienen una breve descripción que ayudan al usuario a entender en qué consisten, y los botones de funcionalidad global (carga y guardado en fichero, y envío de configuración al *hardware*) también son autodescriptivos.
- La disposición de los parámetros es clara y permite distinguirlos entre ellos fácilmente.

Heurísticas de Nielsen

En el diseño propuesto se han seguido las siguientes heurísticas de Nielsen:

- **Visibilidad del estado del sistema** (número 1): la aplicación mantiene al usuario informado sobre estado del sistema mediante los valores configurables, mostrados en la parte izquierda de la pantalla, y mediante el texto situado a la derecha del

botón de volcado de configuración, que informa sobre el estado de sincronización entre la aplicación y el *hardware*.

- **Correspondencia entre el sistema y la vida real** (número 2): El lenguaje utilizado en la aplicación para hablar del protocolo MIDI y de los controles del *hardware* es el mismo que el que figura en el manual del usuario del controlador y en el mundo del *hardware* MIDI y los programas de edición de audio. El modelo gráfico de los controles y su disposición en la parte derecha de la interfaz gráfica se asemeja de forma muy cercana al Electrix Tweaker.

- **Prevención de errores** (número 5): La aplicación avisa al usuario del estado de desconexión del *Tweaker* si corresponde, y muestra mensajes de error en el caso de que se produzca alguna excepción al guardar y cargar ficheros, realizar el volcado de configuración.

- **Reconocer en lugar de recordar** (número 6): La aplicación muestra ayuda contextual, mostrando información sobre todos los parámetros de cada control del Tweaker. Además, cada botón cuenta con un nombre descriptivo para facilitar su comprensión.

- **Estética y diseño minimalista** (número 8): La aplicación cuenta exclusivamente con las funcionalidades necesarias. No dispone de ventana de bienvenida, ventanas flotantes adicionales o elementos decorativos que interfieran con el objetivo del usuario.

- **Ayudar al usuario a reconocer, diagnosticar y recuperarse de los errores** (número 9): Los mensajes de advertencia y error son descriptivos y, en caso de ser posible, ofrecen soluciones al problema que lo causa.

7.2.3. Diseño implementado

El diseño de la interfaz de usuario de la aplicación en el momento de la entrega de este trabajo es la segunda versión, mencionada anteriormente y mostrada en la figura 7.6. A continuación se describen todas las ventanas que la componen, así como algunos de sus componentes.

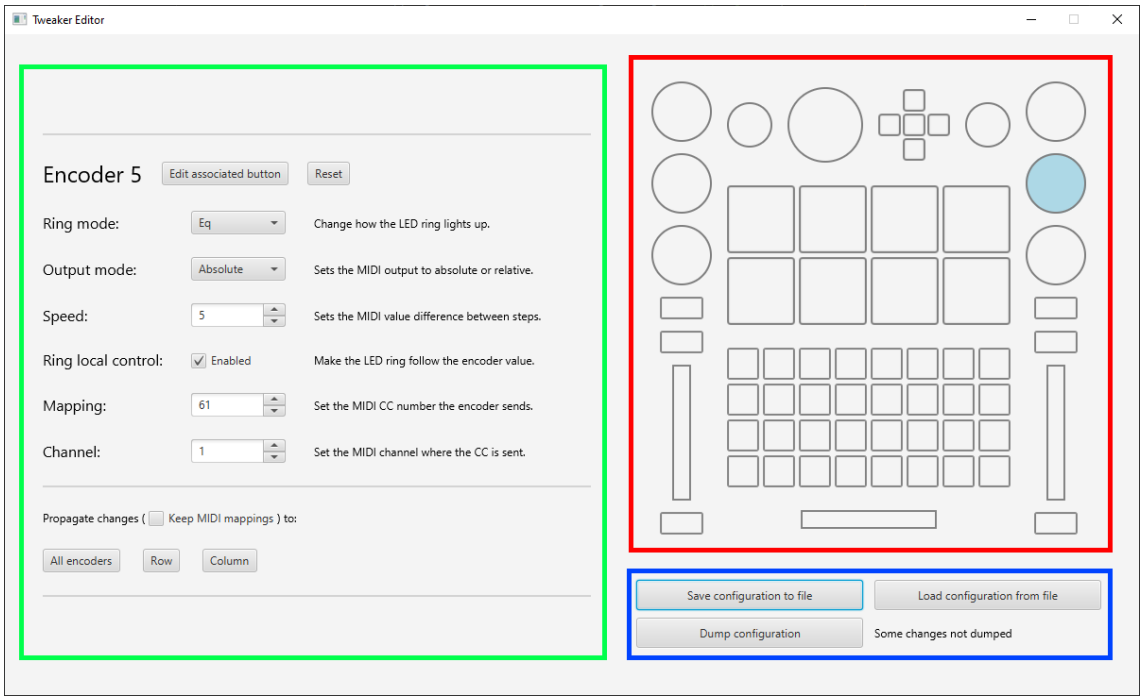


Fig. 7.7. División de secciones de la ventana principal

En la figura 7.7 se muestra gráficamente la distribución de la ventana principal de la interfaz gráfica en tres secciones:

- **Selección de control**, indicada en rojo, permite al usuario elegir el control del *hardware* cuyos parámetros podrá ver y editar.
- **Edición de parámetros**, indicada en verde, donde se muestra el nombre del control seleccionado, así como sus parámetros y las funciones de propagación.
- **Funciones globales**, indicada en azul, con botones que permiten cargar y guardar la configuración en ficheros locales y enviar la configuración al *hardware*.

Las secciones se detallan a continuación.

Selección de control

Permite al usuario elegir el control del Electrix Tweaker cuyos parámetros desea visualizar y/o modificar. Se trata de una visión de la disposición de los controles que refleja con precisión el *hardware* real, por lo que cuenta con un nivel de usabilidad alto. Cada control funciona como un botón, y sólo se puede interactuar con él haciendo clic con el botón izquierdo del ratón.

El control seleccionado se muestra con un resaltado en azul. Ya que sólo puede haber un único control seleccionado al mismo tiempo, al hacer clic en cualquier otro control no seleccionado se cambiará el estado de resaltado y se actualizará la sección de Edición de parámetros.

Edición de parámetros

Está dividida en tres partes, desde arriba hacia abajo:

1. **Nombre** del control, así como un botón para acceder a parámetros extra del mismo (si los tiene) y otro para cambiar los valores de los parámetros a los originales.
2. Lista de **parámetros** del control, con el nombre del parámetro, su valor en un campo editable, y una breve descripción.
3. Funciones de **propagación**, con botones para acceder a los distintos modos y una casilla de verificación para activar el modificador.

En este caso, se cuenta con componentes de interfaz de distinto tipo:

- **Botones**, para acceder a la propagación y otras funciones auxiliares.
- ***Spinners*** o campos númericos, que sólo permiten introducir números enteros en un rango definido, mediante el teclado o haciendo clic sobre las flechas del campo.
- **Casillas de verificación**, que permiten marcar un campo como verdadero o falso.
- **Desplegables**, que permiten elegir un valor de un conjunto definido.

Funciones globales

Dispone de tres botones para gestionar la carga y guardado de la configuración en ficheros locales, y enviar la configuración actual al Electrix Tweaker. También muestra el estado de la configuración, indicando si se ha realizado algún cambio desde el último envío al *hardware*.

Cuenta únicamente con componentes tipo botón, y tiene más elementos dinámicos aparte del texto de estado.

Otras ventanas

Durante el uso de la aplicación se pueden presentar otras disposiciones y ventanas de la interfaz gráfica. Estas son las siguientes:

- Al inicio de la aplicación, dado que ningún control ha sido seleccionado, la sección de Edición de parámetros muestra un mensaje de ayuda para el usuario. Esta situación se muestra en la figura 7.8.
- Al cargar o guardar la configuración en un fichero local, la aplicación muestra una ventana bloqueante nueva con un buscador de ficheros, que permite al usuario localizar una ruta para operar con ella (visible en la figura 7.9).
- En caso de que la aplicación quiera informar al usuario de un estado que pueda generar problemas, se abre una ventana bloqueante con el símbolo de advertencia, como se muestra en el caso de la figura 7.10.
- Si la aplicación se encuentra con un error, tanto interno como causado por el *hardware* o la gestión de ficheros, se abre una ventana bloqueante con el símbolo de error y el mensaje en cuestión, como se observa en la figura 7.11.

7.3. Fichero de configuración

El archivo de configuración que puede generar y cargar la aplicación sigue internamente el formato JSON. En el caso de esta aplicación, al pedir al usuario la ruta y el nombre del fichero donde guardar la configuración actual, en la ventana de buscador de ficheros, se permite seleccionar el tipo de fichero `.json` o `.twp`. El formato `.twp` (de TWeaker Patch, o configuración de Tweaker) es simplemente un nombre alternativo para el formato `.json`, un fichero de texto.

Se ha decidido usar este formato para guardar las configuraciones por dos motivos principales:

1. Se trata de un formato abierto y fácilmente legible. Esto permite tanto crear utilidades y programas que lean y/o modifiquen estos ficheros, como que los usuarios

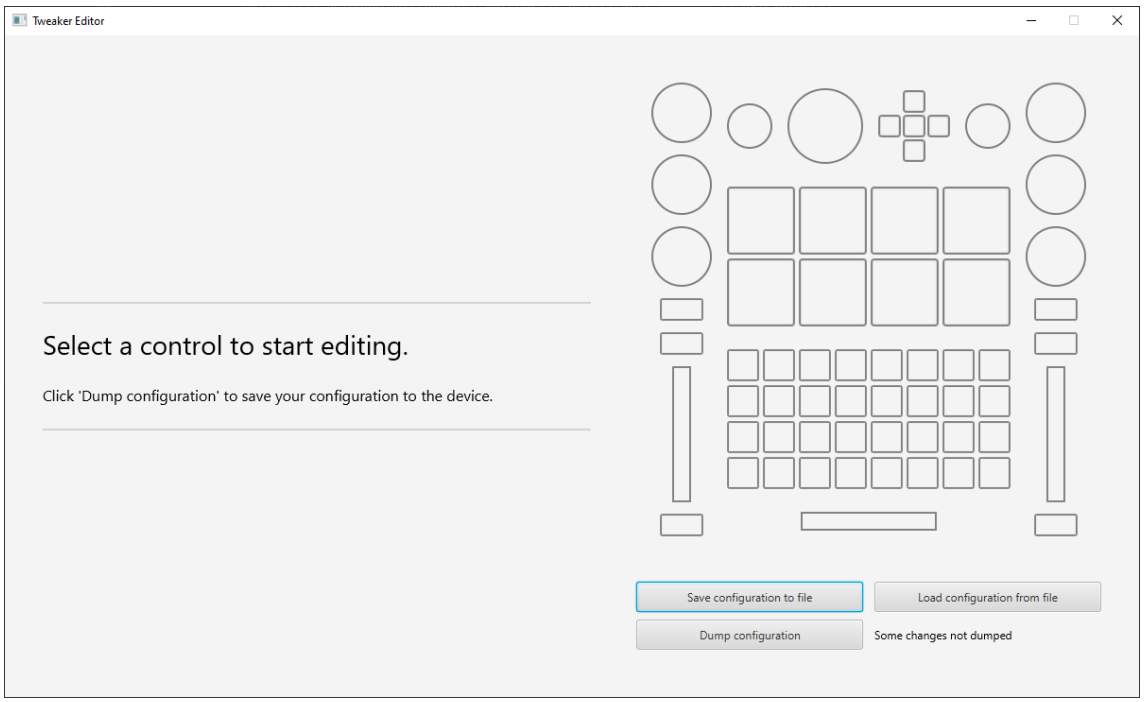


Fig. 7.8. Estado inicial de la ventana principal

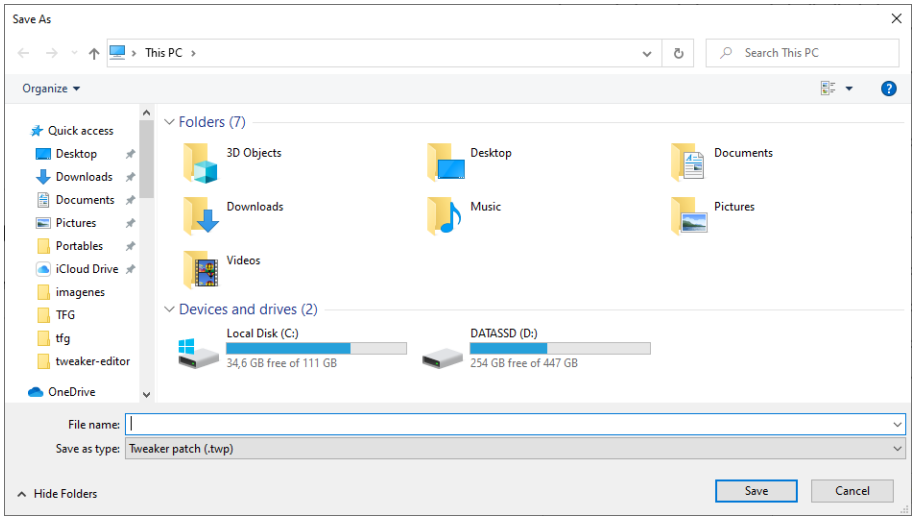


Fig. 7.9. Ventana de buscador de ficheros en Windows

Fig. 7.10. Ventana de mensaje de advertencia

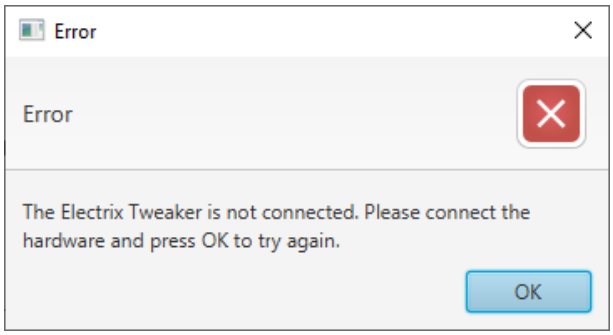


Fig. 7.11. Ventana de mensaje de error

editen el fichero directamente con un programa editor de texto como Notepad, VIM o Sublime Text.

2. Es uno de los formatos de representación de datos más usados, existiendo multitud de herramientas y bibliotecas para gestionarlos, y cuenta con soporte nativo en muchos lenguajes de programación y *scripting* como JavaScript, Java, Python, Go y otros.

El formato de los datos del fichero que genera la aplicación es el siguiente:

- `creation_date`: Fecha de generación del fichero.
- `encoders`: Lista de objetos, donde cada elemento corresponde con un codificador rotativo diferente según su posición en la lista (ver figura 3.1 con las correspondencias entre control e identificador/índice). Cada objeto contiene los parámetros del codificador correspondiente.
- `potentiometers`: Lista de objetos, en este caso, representando los potenciómetros lineales y rotativos.
- `buttons`: Lista con los parámetros de los botones.
- `pads`: Lista con los parámetros de los *pads*.
- `butLeds`: Lista con los parámetros de los LEDs de los botones, excepto los de navegación.
- `navLeds`: Lista con los parámetros de los LEDs de los botones de navegación.

En el momento de cargar el fichero como configuración con el prototipo se realiza una verificación de formato, asegurándose de que al menos los parámetros de los controles existen y tienen valores válidos (dentro de los rangos aceptables). Otros campos que puedan existir en el fichero, como `creation_date` (generado en el guardado) son ignorados automáticamente.

7.4. Distribución

Para que la aplicación desarrollada en este proyecto llegue a los usuarios finales se debe distribuir un binario o un instalador que permita ejecutarla de forma sencilla en cualquier ordenador compatible. Para ello, en esta sección se analiza la forma de compilar y empaquetar la aplicación que nos permite distribuirla de la forma más conveniente posible para el usuario.

Al tratarse de un programa basado en Java, dependiente además de bibliotecas de terceras partes (JavaFX y Gson), se plantean una serie de cuestiones que se deben resolver de la mejor manera posible para mejorar la usabilidad y facilidad de acceso del proyecto.

Por una parte, al estar basada en Java, la aplicación requiere de un Entorno de Ejecución de Java (Java Runtime Environment, o JRE) para funcionar. Este puede estar instalado de forma global en el sistema, de tal forma que cualquier aplicación Java utilice el mismo entorno, o puede existir dentro del contexto de una única aplicación y no ser compartido con ninguna otra (un JRE personalizado).

Un JRE global facilita aplicar actualizaciones de seguridad, evitando posibles problemas en cualquier aplicación Java. Asimismo, Un JRE personalizado para cada aplicación supone un esfuerzo extra a la hora de aplicar estas actualizaciones, ya que cada distribuidor deberá trabajar en la actualización por su cuenta. Además supone un uso del almacenamiento superior, dado que existe una copia del entorno por cada aplicación.

Por otro lado, un JRE personalizado permite tener un control más específico sobre la versión del entorno y las bibliotecas que se incluyen en él, pudiendo potencialmente reducir su tamaño para adecuarse exclusivamente a la aplicación a la que se dedica excluyendo funcionalidades que no se usan e integrando funcionalidades extra que no están incluidas en un entorno global por defecto, como JavaFX o Gson.

Para este proyecto se ha escogido realizar un empaquetamiento del JRE con la aplicación, usando la utilidad jpackage.

7.4.1. jpackage

Se trata de una herramienta incluida en el Kit de Desarrollo de Java (JDK) desde la versión 14 que permite empaquetar una aplicación de Java (`.jar`) y un JRE en un instalador o imagen (dependiendo del sistema operativo) que incluye todas las dependencias necesarias [38].

Esta utilidad permite generar instaladores para los tres sistemas operativos principales (Windows, macOS y Linux). En este caso, se ha optado por generar únicamente el instalador para Windows ya que es el sistema operativo del que se dispone en el entorno de desarrollo. Para que la generación se lleve a cabo es necesario, en el caso de Windows, disponer de WiX en el entorno.

Se generar el paquete utilizando un único comando de consola, en este caso:

```
jpackage --input pack/ --module-path mods/
--add-modules javafx.controls,javafx.fxml
--name "Tweaker Editor" --app-version 0.1
--vendor "Jorge Marcos Chávez"
--main-jar tweaker-editor.jar
--license-file LICENSE.md --icon img/icon.ico
--win-menu --win-dir-chooser
```

Los argumentos del comando se explican a continuación:

- `input`: Ruta de la carpeta con los archivos a empaquetar. Debe contener la aplicación (`.jar`), y puede contener otros archivos, como el `README.md` o el `LICENSE.md`.
- `module-path`: Ruta de la carpeta con los módulos de la aplicación que se desean incluir (bibliotecas). En este caso, en la carpeta `mods/` se encuentran los módulos de JavaFX.
- `add-modules`: Selección de módulos a importar. Incluyendo únicamente `javafx.controls`

y `javafx.fxml` se obtiene toda la funcionalidad necesaria mediante la resolución de dependencias.

- `name`: Nombre de la aplicación, por el momento "Tweaker Editor".
- `app-version`: Número de versión de la aplicación.
- `vendor`: Fabricante o desarrollador.
- `main-jar`: Paquete de la aplicación en formato `.jar`, Java ARchive, que debe de existir en la carpeta indicada en el argumento `input`.
- `license-file`: Archivo de texto con la licencia mostrada al inicio del instalador, que el usuario deberá aceptar para continuar.
- `icon`: Icono de la aplicación. Se mostrará, una vez instalado, en accesos directos, barra de búsqueda y como icono de la aplicación en ejecución.
- `win-menu`: Bandera que indica al instalador que deberá crear un enlace a la aplicación en el menú de inicio de Windows.
- `win-dir-chooser`: Bandera que indica al instalador que deberá permitir al usuario elegir la ruta de instalación mostrando un buscador de archivos.

7.4.2. Resultados

Utilizando la utilidad jpackage con el comando visto en la sección anterior, se obtiene un instalador de la aplicación para Windows. En la figura 7.12 se muestra una pantalla del instalador, donde aparece la licencia del *software*.

Para la distribución de la aplicación se ha diseñado un icono simple y reconocible, mostrado en la figura 7.13.

Tras completar la instalación, la aplicación aparece en el menú de inicio de Windows, como indicaba la bandera utilizada en el comando con jpackage de generación del paquete. Se puede ver la aplicación instalada en la figura 7.14.

Ya que no se dispone de capacidades para firmar el paquete y la aplicación en el entorno de desarrollo utilizado en el proyecto, el sistema operativo Windows (y potencialmente macOS) muestra alertas avisando de la falta de verificación del instalador. En

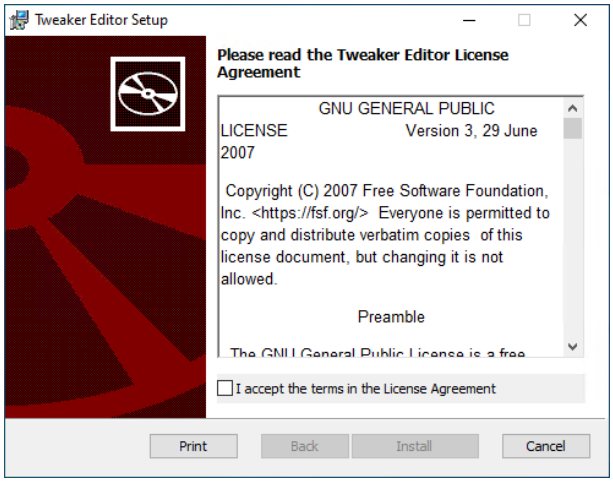


Fig. 7.12. Instalador de la aplicación en Windows

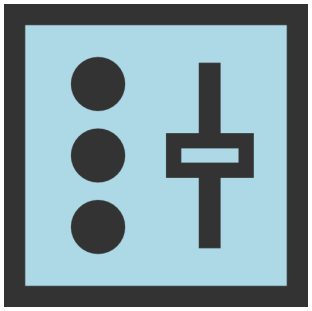


Fig. 7.13. Icono diseñado para la aplicación

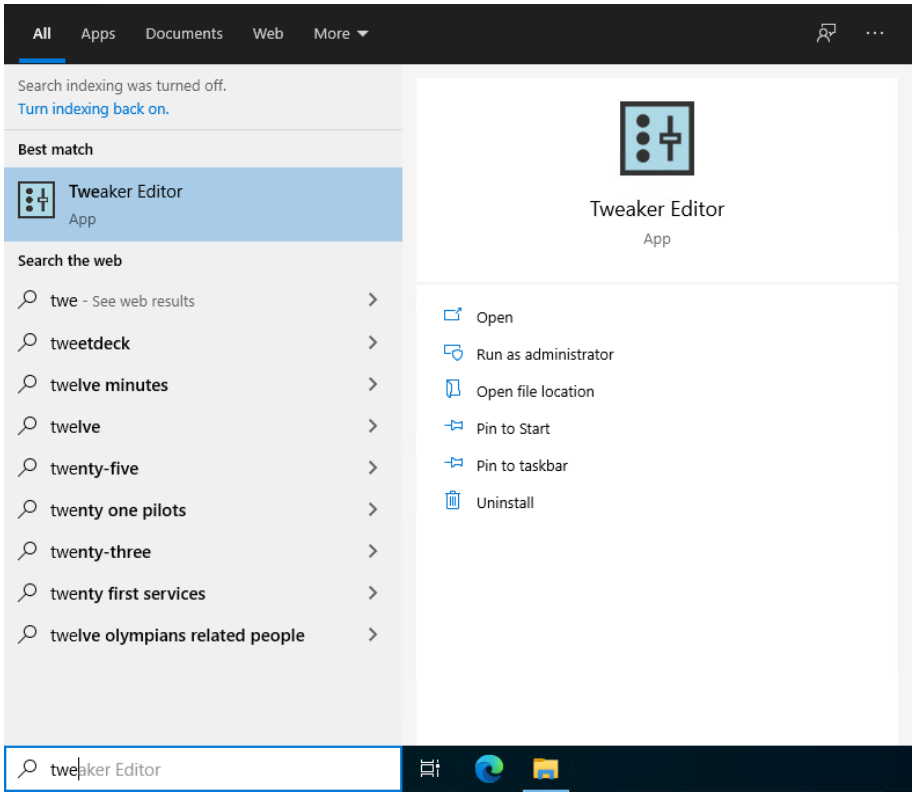


Fig. 7.14. Aplicación indexada por Windows en el menú de inicio

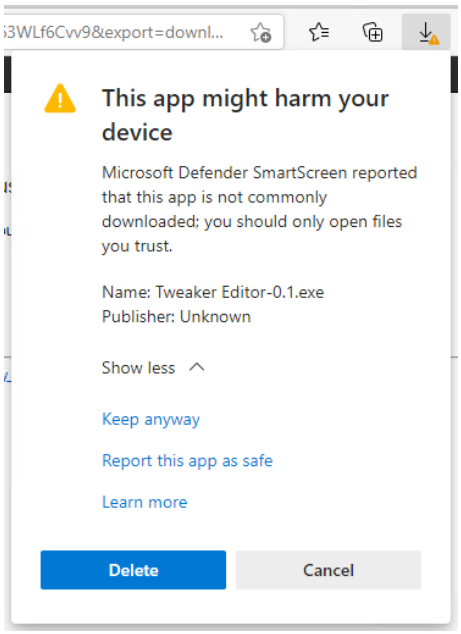


Fig. 7.15. Ejemplo de alerta generada por falta de firma digital

la figura 7.15 se puede apreciar la alerta en el navegador Microsoft Edge cuando se intenta descargar el paquete. Una alerta similar muestra el sistema SmartScreen de Windows al ejecutar el instalador.

Esto se puede solucionar obteniendo un certificado de firmado, distribuido por el fabricante del sistema operativo, y utilizando una herramienta de firma. En este caso, jpackage proporciona la funcionalidad necesaria para firmar el paquete completo.

7.4.3. Otras plataformas

Debido a la falta de soporte para los sistemas operativos macOS y Linux en el entorno de desarrollo del proyecto, los paquetes para estas dos plataformas no pueden ser generados por el momento. A pesar de ello, se cuenta con el soporte adecuado para llevarlo a cabo en el futuro [39].

La opción más económica para desplegar un entorno de distribución de la aplicación es la creación de máquinas virtuales que permitan la compilación y el empaquetamiento en los tres sistemas operativos utilizando una única máquina. Debido a que no es posible virtualizar macOS de forma legal (a no ser que se haga sobre un anfitrión con el mismo sistema operativo), se debería optar por construir el entorno de distribución sobre éste, con máquinas virtuales Windows y Linux.

Esto supone un problema adicional, y es que con el cambio de arquitectura de procesamiento de los ordenadores de Apple de x86 a ARM se presenta un problema de longevidad del *software*:

- Si se escoge la arquitectura x86, se pueden crear máquinas virtuales Windows y Linux con la misma arquitectura y se podrán generar paquetes para los tres sistemas operativos. Los paquetes para macOS, sin embargo, están destinados a quedar obsoletos en pocos años ya que esta arquitectura está siendo reemplazada rápidamente en este sistema operativo.
- Si se escoge ARM, la compatibilidad futura con sistemas de Apple es más probable pero no se pueden generar paquetes para arquitecturas x86 en Windows o Linux ya que jpackage no dispone de capacidades de compilación cruzada.

Por tanto, para asegurar la disponibilidad de esta aplicación en todos los sistemas operativos y arquitecturas, la mejor opción es disponer de dos máquinas: Un sistema Apple con arquitectura ARM, y otro Windows (con virtualización de Linux) con arquitectura x86.

8. PRUEBAS

Para la verificación del funcionamiento de este proyecto se han diseñado pruebas de dos tipos:

- Las **pruebas unitarias** evalúan el funcionamiento interno de la aplicación exclusivamente.
- Las **pruebas de usabilidad** evalúan tanto el diseño de la experiencia de usuario y la interfaz gráfica, como el funcionamiento de la aplicación internamente (verificación de casos de uso y requisitos).

8.1. Pruebas unitarias

Las pruebas unitarias tienen como objetivo verificar el funcionamiento correcto de los componentes internos de la aplicación de forma individual. En el caso de este proyecto, se ha escogido desarrollar este tipo de pruebas para el componente `TweakerConfig`, el modelo de datos, ya que es el único componente con suficientes métodos que no interaccionan con el *hardware* o con la interfaz gráfica.

El formato de la definición del caso de prueba unitaria es el siguiente:

ID	Método	Argumento(s)	Resultado esperado
PU-XXX	`methodName`	-	-

TABLA 8.1. FORMATO DE LA TABLA DE CASOS DE PRUEBA UNITARIA

La definición de cada campo de la tabla anterior es:

- **ID**: Identificador único del caso de prueba, que comienza con PU (Prueba Unitaria) y contiene a continuación el descriptor numérico, separado por un guión.
- **Método**: Método de la clase `TweakerConfig` sobre el que se evalúa en la prueba actual.

- **Argumento o argumentos**: Valores introducidos como argumentos en el método a probar.
- **Resultado esperado**: Cambios en el sistema o excepciones generadas por el método que deben resultar de su ejecución para que la prueba sea válida. Los resultados posibles son:
    - Funcionamiento correcto: El método devuelve un valor dependiente de la configuración en el momento de su ejecución en el caso de ser de tipo `get`, o no genera excepción en el caso de ser de tipo `set`.
    - Excepción: El método genera una excepción de tipo *IllegalArgumentException* (argumento no válido) al recibir una combinación de argumentos inválida.

A continuación se definen los casos de prueba unitaria diseñados para los métodos de la clase `TweakerConfig`:

ID	Método	Argumento(s)	Resultado esperado
PU-001	`encGetRingMode`	Válido	Funcionamiento correcto
PU-002	`encGetRingMode`	*Id* incorrecto	Excepción
PU-003	`encSetRingMode`	Ambos válidos	Funcionamiento correcto
PU-004	`encSetRingMode`	*Id* incorrecto	Excepción
PU-005	`encSetRingMode`	*Mode* incorrecto	Excepción
PU-006	`encSetRingMode`	Ambos incorrectos	Excepción
PU-007	`encGetRelativeMode`	Válido	Funcionamiento correcto
PU-008	`encGetRelativeMode`	*Id* incorrecto	Excepción
PU-009	`encSetRelativeMode`	Ambos válidos	Funcionamiento correcto
PU-010	`encSetRelativeMode`	*Id* incorrecto	Excepción
PU-011	`encSetRelativeMode`	*Mode* incorrecto	Excepción
PU-012	`encSetRelativeMode`	Ambos incorrectos	Excepción
PU-013	`encGetSpeed`	Válido	Funcionamiento correcto
PU-014	`encGetSpeed`	*Id* incorrecto	Excepción
PU-015	`encSetSpeed`	Ambos válidos	Funcionamiento correcto
PU-016	`encSetSpeed`	*Id* incorrecto	Excepción
PU-017	`encSetSpeed`	*Speed* incorrecto	Excepción
PU-018	`encSetSpeed`	Ambos incorrectos	Excepción
PU-019	`encGetLocalControl`	Válido	Funcionamiento correcto
PU-020	`encGetLocalControl`	*Id* incorrecto	Excepción
PU-021	`encSetLocalControl`	Ambos válidos	Funcionamiento correcto
PU-022	`encSetLocalControl`	*Id* incorrecto	Excepción
PU-023	`encSetLocalControl`	*LocalControl* incorrecto	Excepción
PU-024	`encSetLocalControl`	Ambos incorrectos	Excepción
PU-025	`encGetMapping`	Válido	Funcionamiento correcto

TABLA 8.2. CASOS DE PRUEBA UNITARIA 1 A 25

ID	Método	Argumento(s)	Resultado esperado
PU-026	`encGetMapping`	*Id* incorrecto	Excepción
PU-027	`encSetMapping`	Ambos válidos	Funcionamiento correcto
PU-028	`encSetMapping`	*Id* incorrecto	Excepción
PU-029	`encSetMapping`	*Mapping* incorrecto	Excepción
PU-030	`encSetMapping`	Ambos incorrectos	Excepción
PU-031	`encGetChannel`	Válido	Funcionamiento correcto
PU-032	`encGetChannel`	*Id* incorrecto	Excepción
PU-033	`encSetChannel`	Ambos válidos	Funcionamiento correcto
PU-034	`encSetChannel`	*Id* incorrecto	Excepción
PU-035	`encSetChannel`	*Channel* incorrecto	Excepción
PU-036	`encSetChannel`	Ambos incorrectos	Excepción
PU-037	`potGetMapping`	Válido	Funcionamiento correcto
PU-038	`potGetMapping`	*Id* incorrecto	Excepción
PU-039	`potSetMapping`	Ambos válidos	Funcionamiento correcto
PU-040	`potSetMapping`	*Id* incorrecto	Excepción
PU-041	`potSetMapping`	*Mapping* incorrecto	Excepción
PU-042	`potSetMapping`	Ambos incorrectos	Excepción
PU-043	`potGetChannel`	Válido	Funcionamiento correcto
PU-044	`potGetChannel`	*Id* incorrecto	Excepción
PU-045	`potSetChannel`	Ambos válidos	Funcionamiento correcto
PU-046	`potSetChannel`	*Id* incorrecto	Excepción
PU-047	`potSetChannel`	*Channel* incorrecto	Excepción
PU-048	`potSetChannel`	Ambos incorrectos	Excepción
PU-049	`butGetMapping`	Válido	Funcionamiento correcto
PU-050	`butGetMapping`	*Id* incorrecto	Excepción

TABLA 8.3. CASOS DE PRUEBA UNITARIA 25 A 50

ID	Método	Argumento(s)	Resultado esperado
PU-051	`butSetMapping`	Ambos válidos	Funcionamiento correcto
PU-052	`butSetMapping`	*Id* incorrecto	Excepción
PU-053	`butSetMapping`	*Mapping* incorrecto	Excepción
PU-054	`butSetMapping`	Ambos incorrectos	Excepción
PU-055	`butGetChannel`	Válido	Funcionamiento correcto
PU-056	`butGetChannel`	*Id* incorrecto	Excepción
PU-057	`butSetChannel`	Ambos válidos	Funcionamiento correcto
PU-058	`butSetChannel`	*Id* incorrecto	Excepción
PU-059	`butSetChannel`	*Channel* incorrecto	Excepción
PU-060	`butSetChannel`	Ambos incorrectos	Excepción
PU-061	`butGetOutputType`	Válido	Funcionamiento correcto
PU-062	`butGetOutputType`	*Id* incorrecto	Excepción
PU-063	`butSetOutputType`	Ambos válidos	Funcionamiento correcto
PU-064	`butSetOutputType`	*Id* incorrecto	Excepción
PU-065	`butSetOutputType`	*Type* incorrecto	Excepción
PU-066	`butSetOutputType`	Ambos incorrectos	Excepción
PU-067	`butGetSpeedControl`	Válido	Funcionamiento correcto
PU-068	`butGetSpeedControl`	*Id* incorrecto	Excepción
PU-069	`butSetSpeedControl`	Ambos válidos	Funcionamiento correcto
PU-070	`butSetSpeedControl`	*Id* incorrecto	Excepción
PU-071	`butSetSpeedControl`	*Control* incorrecto	Excepción
PU-072	`butSetSpeedControl`	Ambos incorrectos	Excepción
PU-073	`butGetLocalControl`	Válido	Funcionamiento correcto
PU-074	`butGetLocalControl`	*Id* incorrecto	Excepción
PU-075	`butSetLocalControl`	Ambos válidos	Funcionamiento correcto

TABLA 8.4. CASOS DE PRUEBA UNITARIA 51 A 75

ID	Método	Argumento(s)	Resultado esperado
PU-076	`butSetLocalControl`	*Id* incorrecto	Excepción
PU-077	`butSetLocalControl`	*Control* incorrecto	Excepción
PU-078	`butSetLocalControl`	Ambos incorrectos	Excepción
PU-079	`padGetHitMapping`	Válido	Funcionamiento correcto
PU-080	`padGetHitMapping`	*Id* incorrecto	Excepción
PU-081	`padSetHitMapping`	Ambos válidos	Funcionamiento correcto
PU-082	`padSetHitMapping`	*Id* incorrecto	Excepción
PU-083	`padSetHitMapping`	*Mapping* incorrecto	Excepción
PU-084	`padSetHitMapping`	Ambos incorrectos	Excepción
PU-085	`padGetHitChannel`	Válido	Funcionamiento correcto
PU-086	`padGetHitChannel`	*Id* incorrecto	Excepción
PU-087	`padSetHitChannel`	Ambos válidos	Funcionamiento correcto
PU-088	`padSetHitChannel`	*Id* incorrecto	Excepción
PU-089	`padSetHitChannel`	*Channel* incorrecto	Excepción
PU-090	`padSetHitChannel`	Ambos incorrectos	Excepción
PU-091	`padGetRetriggerMapping`	Válido	Funcionamiento correcto
PU-092	`padGetRetriggerMapping`	*Id* incorrecto	Excepción
PU-093	`padSetRetriggerMapping`	Ambos válidos	Funcionamiento correcto
PU-094	`padSetRetriggerMapping`	*Id* incorrecto	Excepción
PU-095	`padSetRetriggerMapping`	*Mapping* incorrecto	Excepción
PU-096	`padSetRetriggerMapping`	Ambos incorrectos	Excepción
PU-097	`padGetRetriggerChannel`	Válido	Funcionamiento correcto
PU-098	`padGetRetriggerChannel`	*Id* incorrecto	Excepción
PU-099	`padSetRetriggerChannel`	Ambos válidos	Funcionamiento correcto
PU-100	`padSetRetriggerChannel`	*Id* incorrecto	Excepción

TABLA 8.5. CASOS DE PRUEBA UNITARIA 76 A 100

ID	Método	Argumento(s)	Resultado esperado
PU-101	`padSetRetriggerChannel`	*Channel* incorrecto	Excepción
PU-102	`padSetRetriggerChannel`	Ambos incorrectos	Excepción
PU-103	`butLedGetColor`	Válido	Funcionamiento correcto
PU-104	`butLedGetColor`	*Id* incorrecto	Excepción
PU-105	`butLedSetColor`	Ambos válidos	Funcionamiento correcto
PU-106	`butLedSetColor`	*Id* incorrecto	Excepción
PU-107	`butLedSetColor`	*Color* incorrecto	Excepción
PU-108	`butLedSetColor`	Ambos incorrectos	Excepción
PU-109	`butLedGetMapping`	Válido	Funcionamiento correcto
PU-110	`butLedGetMapping`	*Id* incorrecto	Excepción
PU-111	`butLedSetMapping`	Ambos válidos	Funcionamiento correcto
PU-112	`butLedSetMapping`	*Id* incorrecto	Excepción
PU-113	`butLedSetMapping`	*Mapping* incorrecto	Excepción
PU-114	`butLedSetMapping`	Ambos incorrectos	Excepción
PU-115	`butLedGetChannel`	Válido	Funcionamiento correcto

TABLA 8.6. CASOS DE PRUEBA UNITARIA 101 A 115

8.2. Pruebas de usabilidad

Las pruebas de usabilidad tienen como objetivo verificar si la experiencia de usuario del sistema es adecuada mediante pruebas reales con potenciales usuarios. Para el propósito de este proyecto, se diseñan casos de prueba de usabilidad basados en los casos de uso definidos en la sección 5.2.

Este tipo de pruebas permiten evaluar tanto el correcto funcionamiento de la funcionalidad de la aplicación como las decisiones de diseño de la interfaz gráfica y la experiencia de usuario, ya que los casos de prueba pueden fallar, es decir, no verificar la funcionalidad de la aplicación, si esta no cuenta con ella o si el usuario es incapaz de interactuar con el sistema para usarla y conseguir su objetivo.

El formato de la definición del caso de prueba de usabilidad es el siguiente:

ID	PDU-XX
Nombre	
Objetivos	
Tiempo esperado	
Pasos esperados	
Casos de uso	

TABLA 8.7. FORMATO DEL CASO DE PRUEBA DE USABILIDAD

La definición de cada campo de la tabla anterior es:

- **ID**: Identificador único del caso de prueba, que comienza con PDU (Prueba De Usabilidad) y contiene a continuación el descriptor numérico, separado por un guión.
- **Nombre**: Breve descripción del caso de prueba de usabilidad.
- **Objetivos**: Cambios en el sistema que se espera que el usuario lleve a cabo.
- **Tiempo esperado**: Tiempo máximo aproximado del que dispone el usuario para completar el caso de prueba satisfactoriamente.
- **Pasos esperados**: Tareas que el usuario debe realizar para completar el caso de prueba satisfactoriamente, explicadas de forma general.
- **Casos de uso**: Casos de uso relacionados con el caso de prueba (verificados por este caso de prueba).

8.2.1. Definición de casos de prueba

Los casos de prueba de usabilidad diseñados para este proyecto son los siguientes:

ID	PDU-01
Nombre	Modificación de configuración y volcado al *hardware* con conexión previa
Objetivos	Cambio de los valores de los parámetros del Tweaker para coincidir con los mostrados en la interfaz gráfica de la aplicación.
Tiempo esperado	30s - 5min (dependiendo de la cantidad de modificaciones)
Pasos esperados	1. Se realiza la conexión física del Tweaker al ordenador mediante un cable USB. 2. Se ejecuta la aplicación. 3. Se modifica uno o varios parámetros de la configuración inicial usando la interfaz gráfica. 4. Se vuelca la configuración al Tweaker. 5. Se cierra la aplicación.
Casos de uso	CU-01, CU-02

TABLA 8.8. CASO DE PRUEBA DE USABILIDAD 1

ID	PDU-02
Nombre	Modificación de configuración de fichero local
Objetivos	Carga, modificación y guardado de la configuración almacenada en un fichero local.
Tiempo esperado	30s - 5min (dependiendo de la cantidad de modificaciones)
Pasos esperados	1. Se ejecuta la aplicación. 2. Se carga la configuración desde un fichero local. 3. Se modifica la configuración usando la interfaz gráfica. 4. Se guarda la configuración en un fichero local.
Casos de uso	CU-01, CU-03, CU-04

TABLA 8.9. CASO DE PRUEBA DE USABILIDAD 2

ID	PDU-03
Nombre	Volcado de configuración desde un fichero local al Tweaker con conexión posterior al inicio
Objetivos	Volcado de configuración desde un fichero local conectando el Tweaker después de ejecutar la aplicación.
Tiempo esperado	30s
Pasos esperados	1. Se ejecuta la aplicación. 2. Se carga la configuración desde un fichero local. 3. Se realiza la conexión física del Tweaker al ordenador mediante un cable USB. 4. Se intenta volcar la configuración al Tweaker, mostrándose el diálogo de reconexión antes de producirse el volcado. 4. Se realiza la reconexión y se termina el volcado.
Casos de uso	CU-02, CU-03

TABLA 8.10. CASO DE PRUEBA DE USABILIDAD 3

ID	PDU-04
Nombre	Intento de volcado sin conexión al *hardware*
Objetivos	Disparo del intento de reconexión al volcar sin la conexión del Tweaker establecida, y fallo de dicha operación.
Tiempo esperado	15s
Pasos esperados	1. Se ejecuta la aplicación. 2. Se intenta volcar la configuración al Tweaker, mostrándose el diálogo de reconexión. 3. Se intenta realizar la reconexión y falla, cancelándose el volcado. 4. Se realiza una acción cualquiera sobre la aplicación, como la modificación de un parámetro.
Casos de uso	CU-02

TABLA 8.11. CASO DE PRUEBA DE USABILIDAD 4

ID	PDU-05
Nombre	Propagación de parámetros de un botón en fila, incluyendo los mapeos
Objetivos	Modificación de la configuración interna de varios botones simultáneamente usando una propagación en fila.
Tiempo esperado	30s - 60s (dependiendo de la cantidad de modificaciones)
Pasos esperados	1. Se ejecuta la aplicación. 2. Se selecciona uno de los botones de la cuadrícula y se modifica uno o varios parámetros. 3. Se realiza la propagación en fila haciendo clic en el botón correspondiente de la interfaz gráfica. 4. Se comprueba que los parámetros del resto de botones de la fila de la cuadrícula han sido modificados.
Casos de uso	CU-01, CU-05

TABLA 8.12. CASO DE PRUEBA DE USABILIDAD 5

ID	PDU-06
Nombre	Propagación de parámetros de un codificador rotativo en columna, manteniendo los mapeos originales
Objetivos	Modificación de la configuración interna de varios codificadores rotativos simultáneamente usando una propagación en columna, manteniendo los mapeos originales.
Tiempo esperado	30s - 60s (dependiendo de la cantidad de modificaciones)
Pasos esperados	1. Se ejecuta la aplicación. 2. Se selecciona uno de los codificadores rotativos con anillo LED y se modifica uno o varios parámetros. 3. Se realiza la propagación en columna haciendo clic en el botón correspondiente de la interfaz gráfica. 4. Se comprueba que los parámetros del resto de codificadores rotativos de la columna han sido modificados a excepción de los mapeos y canales MIDI.
Casos de uso	CU-01, CU-05

TABLA 8.13. CASO DE PRUEBA DE USABILIDAD 6

8.2.2. Matriz de trazabilidad

En la siguiente matriz de trazabilidad se relacionan los casos de prueba de usabilidad con los casos de uso definidos en la sección 5.2:

	CU-01	**CU-02**	**CU-03**	**CU-04**	**CU-05**
PDU-01	O	O			
PDU-02	O		O	O	
PDU-03		O	O		
PDU-04		O			
PDU-05	O				O
PDU-06	O				O

TABLA 8.14. MATRIZ DE TRAZABILIDAD DE CASOS DE PRUEBA DE USABILIDAD Y CASOS DE USO

Como se puede observar, las pruebas de usabilidad cubren todos los casos de uso definidos en el sistema.

8.3. Evaluación de los resultados

A continuación se analizan los resultados de las pruebas de usabilidad y se presentan las conclusiones sobre la experiencia de usuario del prototipo de la aplicación implementada. Por restricciones de tiempo, no se ha considerado la realización de las pruebas unitarias ya que se han considerado de menor relevancia.

8.3.1. Resultados de usabilidad

Se ha contado con 6 sujetos de prueba, con diversos niveles de experiencia en dos campos relevantes para el proyecto: manejo de herramientas informáticas dado el estudio de un grado en ingeniería, y experiencia como músico y con *hardware* MIDI.

Se pueden agrupar estos sujetos de prueba en tres grupos:

- Personas sin estudios de grado en ingeniería y sin experiencia musical o con *hardware* MIDI: SP-01 y SP-02.
- Personas con estudios de grado en ingeniería pero sin experiencia musical o con *hardware* MIDI: SP-03 y SP-04.
- Personas sin estudios de grado en ingeniería pero con experiencia musical y/o con *hardware* MIDI: SP-05 y SP-06.

A continuación se especifican los perfiles de los sujetos, utilizando sólo los campos considerados relevantes para estas pruebas:

Sujeto	SP-01
Ha estudiado una carrera de ingeniería	No
Es músico o trabaja en industria musical	No
Experiencia con *software* musical	Ninguna
Experiencia con controladores MIDI	Ninguna
Experiencia con editores de *hardware* MIDI	Ninguna

TABLA 8.15. DESCRIPCIÓN DEL SUJETO DE PRUEBA SP-01

Sujeto	SP-02
Ha estudiado una carrera de ingeniería	No
Es músico o trabaja en industria musical	No
Experiencia con *software* musical	Ninguna
Experiencia con controladores MIDI	Ninguna
Experiencia con editores de *hardware* MIDI	Ninguna

TABLA 8.16. DESCRIPCIÓN DEL SUJETO DE PRUEBA SP-02

Sujeto	SP-03
Ha estudiado una carrera de ingeniería	Sí
Es músico o trabaja en industria musical	No
Experiencia con *software* musical	Ninguna
Experiencia con controladores MIDI	Ninguna
Experiencia con editores de *hardware* MIDI	Ninguna

TABLA 8.17. DESCRIPCIÓN DEL SUJETO DE PRUEBA SP-03

Sujeto	SP-04
Ha estudiado una carrera de ingeniería	Sí
Es músico o trabaja en industria musical	No
Experiencia con *software* musical	Ninguna
Experiencia con controladores MIDI	Ninguna
Experiencia con editores de *hardware* MIDI	Ninguna

TABLA 8.18. DESCRIPCIÓN DEL SUJETO DE PRUEBA SP-04

Sujeto	SP-05
Ha estudiado una carrera de ingeniería	No
Es músico o trabaja en industria musical	Sí
Experiencia con *software* musical	2 años
Experiencia con controladores MIDI	Ninguna
Experiencia con editores de *hardware* MIDI	Ninguna

TABLA 8.19. DESCRIPCIÓN DEL SUJETO DE PRUEBA SP-05

Sujeto	SP-06
Ha estudiado una carrera de ingeniería	No
Es músico o trabaja en industria musical	Sí
Experiencia con *software* musical	6 años
Experiencia con controladores MIDI	3 años
Experiencia con editores de *hardware* MIDI	Ninguna

TABLA 8.20. DESCRIPCIÓN DEL SUJETO DE PRUEBA SP-06

Los 6 casos de prueba descritos en la sección 8.2.1 se han realizado con los 6 sujetos, por lo que se han recopilado 36 resultados distintos. Los datos recogidos en cada prueba son los siguientes:

- **Tiempo**: Duración aproximada de la prueba.
- **Pasos extra**: Introducción de tareas o interacciones con el sistema por parte del usuario que no entran dentro de la planificación de la prueba.

- **Observaciones**: Detalles de la realización de la prueba con relevancia para la definición de la usabilidad de la aplicación. En general, problemas o situaciones inesperadas durante el desarrollo de la tarea.
- **Intervención**: Refleja si ha sido necesaria la intervención por parte del responsable durante el desarrollo de la prueba.
- **Objetivo**: Cumplimiento del objetivo propuesto por la prueba.
- **Dificultad**: Dificultad percibida por el sujeto sobre la tarea de la prueba.

Para evitar el exceso de información, se omiten los resultados individuales y se procede a presentar un análisis resumido:

1. Todos los casos de prueba han visto su objetivo cumplido con todos los sujetos, aunque en ocasiones han requerido intervención por parte del responsable para confirmarlo. Esto se ha dado con menos frecuencia en los sujetos SP-03 y SP-04, con estudios de ingeniería, lo que significa que la interacción con la aplicación es más sencilla para personas con esa característica.

2. Todos los casos de prueba han sido marcados por los sujetos como fáciles en dificultad, por lo que se asume que tras aprender a solucionar posibles problemas de interacción con la aplicación, su uso es simple y pueden extrapolar el funcionamiento a distintas casuísticas.

3. Todos los sujetos han tenido problemas en los casos de prueba PDU-05 y PDU-06, donde se requiere realizar una propagación, en el primer caso copiando también los mapeos y en el segundo manteniendo los mapeos originales. El problema se ha dado a la hora de marcar o no la casilla "*Keep MIDI mappings*", que o bien ha sido interpretada como "Propagar incluyendo mapeos" en ambos casos de prueba y, por tanto, usado de la manera inversa a la prevista, o bien ha sido interpretado de una forma en el primer caso y de la otra en el segundo, marcándose o desmarcándose en ambos.

4. Asimismo, todos los sujetos han tenido problemas para identificar cuando una operación iniciada por alguno de los botones de las funciones globales ha terminado

correctamente. La expectativa general es mostrar un cuadro de diálogo, similar a los de advertencia y error vistos en la sección 7.2.3, que lo comunique explícitamente. Los sujetos SP-03, SP-04 y SP-06 tuvieron menos problemas para identificar el final de la operación, en parte ayudados por el mensaje de estado de la sección de Funciones globales que anuncia cuando un volcado ha sido realizado correctamente.

5. Todos los sujetos han tenido problemas al interpretar los errores mostrados al intentar realizar un volcado sin el Electrix Tweaker conectado antes de arrancar la aplicación. A pesar de conectarlo y presionar OK tras leer el primer mensaje y realizarse la conexión con el *software* correctamente, no está claro si realmente se ha conectado y realizado el volcado o no, por lo que la mayoría de los sujetos vuelven a pulsar el botón de volcado a pesar de que éste se ha realizado correctamente.

6. Para sujetos con entendimiento reducido del inglés, la barrera del idioma ha impedido el correcto desarrollo de ciertas pruebas sin la intervención del responsable.

7. El tiempo se ha estimado de forma optimista, aunque en la mayoría de los casos en los que se ha excedido, ha sido fruto de la necesidad de intervención por parte del responsable.

8. Ningún sujeto ha realizado pasos extra en la ejecución de los casos de prueba, a excepción de repetición de pasos debido a no interpretar los mensajes de error correctamente.

Finalmente, se presentan soluciones a los problemas analizados:

1. Inclusión de mensajes de confirmación para las funciones globales de carga de fichero, guardado, y volcado al Tweaker. Tendrán un diseño similar a los descritos en la sección 7.2.3.

3. Renombramiento de la casilla “Keep MIDI mappings” en la parte de propagación de la sección de Edición de parámetros. Se requiere buscar la manera más lógica de mostrar esta funcionalidad, siendo entendible a primera vista sin necesitar una explicación por parte de terceros.

4. De nuevo, inclusión de mensajes de confirmación.

5. Rediseño de la interacción con la aplicación cuando el Electrix Tweaker no está conectado y se intenta realizar un volcado. La expectativa general sobre ella es que, tras el mensaje de error, la aplicación no realiza ninguna acción, por lo que se debe diseñar alrededor del hecho de que el usuario tiende a reintentar la operación tras verlo.

6. Traducción de la aplicación o escritura de documentación en otros idiomas, para facilitar su uso con personas con nivel reducido de inglés.

9. MARCO REGULADOR

En esta sección se lleva a cabo el análisis de cuatro aspectos legales relevantes para este proyecto.

9.1. Protección

El código se ha desarrollado y publicado bajo la licencia GPL-3 (GNU General Public License v3), que permite modificar y redistribuir el código original, y usarlo de forma comercial, pero requiere que el código añadido con una modificación también se adhiera a esta misma licencia [40]. Esto permite aprovechar las ventajas de liberar una aplicación como *software* libre, ya que cualquier usuario puede realizar cambios en el código original para añadir funcionalidades o corregir problemas, y evitar su apropiación exclusiva por parte de terceros.

Al tratarse de una licencia abierta, cualquier persona puede copiar o inspirarse en el código desarrollado en el proyecto. Por tanto, la protección que ofrece no está orientada a mantener la originalidad y propiedad de la idea, algoritmo o aplicación, sino a proteger el derecho de manipulación y distribución del código por las características descritas anteriormente. Estas licencias son adecuadas para este tipo de proyectos, donde la problemática viene dada por aplicar una serie de métodos ya conocidos de una forma específica (en este caso, mandar ciertos mensajes MIDI a un *hardware* determinado) y no por desarrollar una metodología nueva pero aplicable a multitud de casos distintos (donde la copia o imitación es más rentable).

9.2. Patentabilidad

Para este proyecto no se ha contemplado la posibilidad de patentar el producto final. El objetivo es la sustitución de una aplicación propietaria anterior por una de código abierto para hacer el ecosistema del Electrix Tweaker más sostenible a largo plazo, y este tipo de aplicaciones han existido durante décadas.

Por otra parte, patentar esta solución carece de motivación alguna a nivel de protección de propiedad intelectual ya que las herramientas usadas y funcionalidades implementadas son conocidas en la industria y son usadas ampliamente por otros desarrolladores de *software* orientado a MIDI y de editores de *hardware* MIDI.

9.3. Privacidad y seguridad

La aplicación desarrollada en este proyecto no recopila información sobre el usuario o la máquina donde se ejecuta. Además, no hace uso de conexión a Internet en ningún momento, por lo que tampoco puede comunicarse con un servidor externo u otras aplicaciones. Esto es verificable por cualquier usuario ya que el proyecto es de código abierto y se puede consultar en su totalidad. Además, la aplicación funciona únicamente en un ordenador y se comunica con un *hardware* usando el protocolo MIDI, sin manipularlo directamente de ninguna forma que pueda causar un fallo o error a largo plazo: la gestión de los mensajes que recibe es responsabilidad del controlador MIDI.

Por otro lado, la licencia GPL-3 bajo la que se rige exime al desarrollador de la aplicación de cualquier responsabilidad o garantía sobre la misma.

Además, en el desarrollo de las pruebas de usabilidad de este proyecto, se ha cumplido con la Ley Orgánica de Protección de Datos en la recopilación de la actividad y rendimiento de cada sujeto en cada caso de prueba:

- Se tratan los datos con exactitud y confidencialidad, recopilando exclusivamente aquellos con interés para el trabajo. No se recogen datos de carácter personal.
- Se especifica la finalidad del tratamiento de los datos recopilados.
- Los datos serán destruidos tras la finalización de la evaluación de este Trabajo Final de Grado. No existirá ninguna otra copia más que la almacenada por el responsable de las pruebas de usabilidad.
- El sujeto de prueba es totalmente voluntario, pudiendo oponerse a su participación antes de firmar el documento de autorización pertinente.

9.4. Estándares técnicos

No se ha empleado ningún estándar técnico o guía de estilo de código directamente, aunque se han utilizado ciertas directrices del estilo de código de Google para Java [41]. De forma indirecta se ha seguido el estándar MIDI 1.0 a través de la biblioteca Java Sound API [42] y el controlador Electrix Tweaker.

10. IMPACTO SOCIOECONÓMICO

El impacto social y económico de este proyecto está muy focalizado debido a la naturaleza de la herramienta que se desarrolla: su público objetivo está formado por los usuarios del controlador MIDI Electrix Tweaker.

Según el NAMM Global Report de 2016, se vendieron alrededor de 70.000 controladores MIDI orientados al *DJing* en Estados Unidos [43] en 2013, fecha de salida del Electrix Tweaker. De acuerdo a este mismo documento, el 42 % del equipo de audio profesional de todo el mundo se vende a Estados Unidos, por lo que se puede asumir que la venta global de controladores MIDI de *DJing* asciende a alrededor de 167.000 unidades anuales en 2013.

Realizando la misma operación para 2014 y 2015, usando la misma fuente, se obtienen los datos representados en la tabla 10.1. Si se asume que uno de cada 100 controladores MIDI que se vendió entre 2013 y 2015 es un Tweaker, y se añaden unos coeficientes que representen el interés por la compra de este *hardware* a lo largo del tiempo (descartando a partir del año 2016 por descatalogación), se puede estimar el número de unidades vendidas de este controlador MIDI en 3.295. Esto significa que el proyecto tiene un alcance potencial de unos 3.295 usuarios, restando aquellos cuyo *hardware* haya fallado o quienes lo hayan reemplazado.

Si se asume que esta cifra es correcta o cercana a la realidad, se puede apreciar que el mercado de la aplicación es relativamente reducido. A pesar de ello, este proyecto es muy relevante dentro de su público objetivo por dos razones:

- Supone la recuperación de parte de la funcionalidad del Electrix Tweaker. El editor original, que permitía modificar la gran cantidad de parámetros de los que dispone el *hardware* internamente y que forman parte de las características que lo definen (gran configurabilidad y adaptabilidad a multitud de casos de uso), no es funcional a día de hoy, como se puede ver en la sección 1.1.

- Tiene una gran proyección a futuro y será posible mantenerla y actualizarla mientras que la comunidad de usuarios del Tweaker exista. La aplicación se ha liberado bajo

	2013	**2014**	**2015**
Unidades (EE. UU.)	70.000	80.000	87.000
Unidades (global)	166.667	190.476	207.143
Cuota de mercado		1 %	
Unidades (cuota)	1.667	1.905	2.071
Interés	100 %	50 %	25 %
Unidades (Tweaker)	1.667	953	675
Total		**3.295**	

TABLA 10.1. ESTIMACIÓN DE UNIDADES VENDIDAS DEL ELECTRIX TWEAKER

una licencia GPL-3, por lo que cualquier persona puede ver, copiar y editar su código.

En cuanto a la explotación económica del proyecto, no se ha planteado ninguna forma de beneficio económico más allá de posibles donaciones monetarias o contribuciones al código fuente, debido en parte a la voluntad de liberarlo como código abierto y el objetivo de alargar la vida del Electrix Tweaker. Otra posibilidad es el contacto con el fabricante u otras empresas dedicadas a diseñar controladores MIDI para una posible colaboración o contrato laboral.

11. PLANIFICACIÓN

Este proyecto ha sido desarrollado durante 8 meses, desde enero hasta agosto de 2021 y ha consistido en las siguientes tareas, divididas en cinco grandes fases:

1. **Análisis del problema**: Planteamiento de objetivos y análisis del estado del arte.
2. **Especificación**: Diseño y especificación del sistema de forma teórica.
3. **Diseño de UX**: Análisis de aplicaciones similares y competidoras, y planteamiento de una solución de UX.
4. **Implementación**: Desarrollo del código de la solución.
5. **Pruebas**: Evaluación del rendimiento y verificación de la solución frente al diseño teórico inicial.

Para esta planificación se han supuesto 20 horas de trabajo a la semana. En la tabla 11.1 se detallan las tareas de cada fase y se indican tanto la duración como las fechas de comienzo y fin de cada una, en semanas. La semana de comienzo está incluida dentro del tiempo de la tarea, mientras que la semana de fin es la primera que no entra en ese tiempo.

Ya que se ha llevado a cabo todo el trabajo con una única persona asumiendo todos los roles necesarios, las tareas se realizan secuencialmente. El diagrama de Gantt, que muestra la evolución y posición temporal de cada una en función de los meses y semanas de desarrollo, se muestra en la figura 11.1.

Como herramientas de gestión se han utilizado Microsoft Word (apuntes y notas) y Microsoft Excel (planificación, diseño de tablas, apuntes). A nivel de gestión del desarrollo, se ha empleado Git como sistema de control de versiones para controlar los cambios en el código de la aplicación.

Tarea	Duración (h.)	Comienzo (semana)	Fin (semana, no inc.)
Análisis del problema	**100**	**1 (enero)**	**6 (febrero)**
Propuesta de proyecto	20	1 (enero)	2 (enero)
Análisis de *hardware*	40	2 (enero)	4 (enero)
Estado del arte	40	4 (enero)	6 (febrero)
Especificación	**60**	**6 (febrero)**	**9 (marzo)**
Requisitos	20	6 (febrero)	7 (febrero)
Casos de uso	20	7 (febrero)	8 (febrero)
Arquitectura	20	8 (febrero)	9 (marzo)
Diseño de UX	**80**	**9 (marzo)**	**13 (abril)**
Análisis de UX	20	9 (marzo)	10 (marzo)
Diseño de UX	20	10 (marzo)	11 (marzo)
Diseño de UI	40	11 (marzo)	13 (abril)
Implementación	**300**	**13 (abril)**	**28 (julio)**
Interfaz gráfica	80	13 (abril)	17 (mayo)
Modelo de datos	60	17 (mayo)	20 (mayo)
Controlador	80	20 (mayo)	24 (junio)
Comunicación MIDI	80	24 (junio)	28 (julio)
Pruebas	**100**	**28 (julio)**	**33 (septiembre)**
Diseño	40	28 (julio)	30 (agosto)
Ejecución	60	30 (agosto)	33 (septiembre)
Total	**640**	**1 (enero)**	**33 (septiembre)**

TABLA 11.1. DETALLE DE LAS TAREAS A DESARROLLAR EN EL PROYECTO

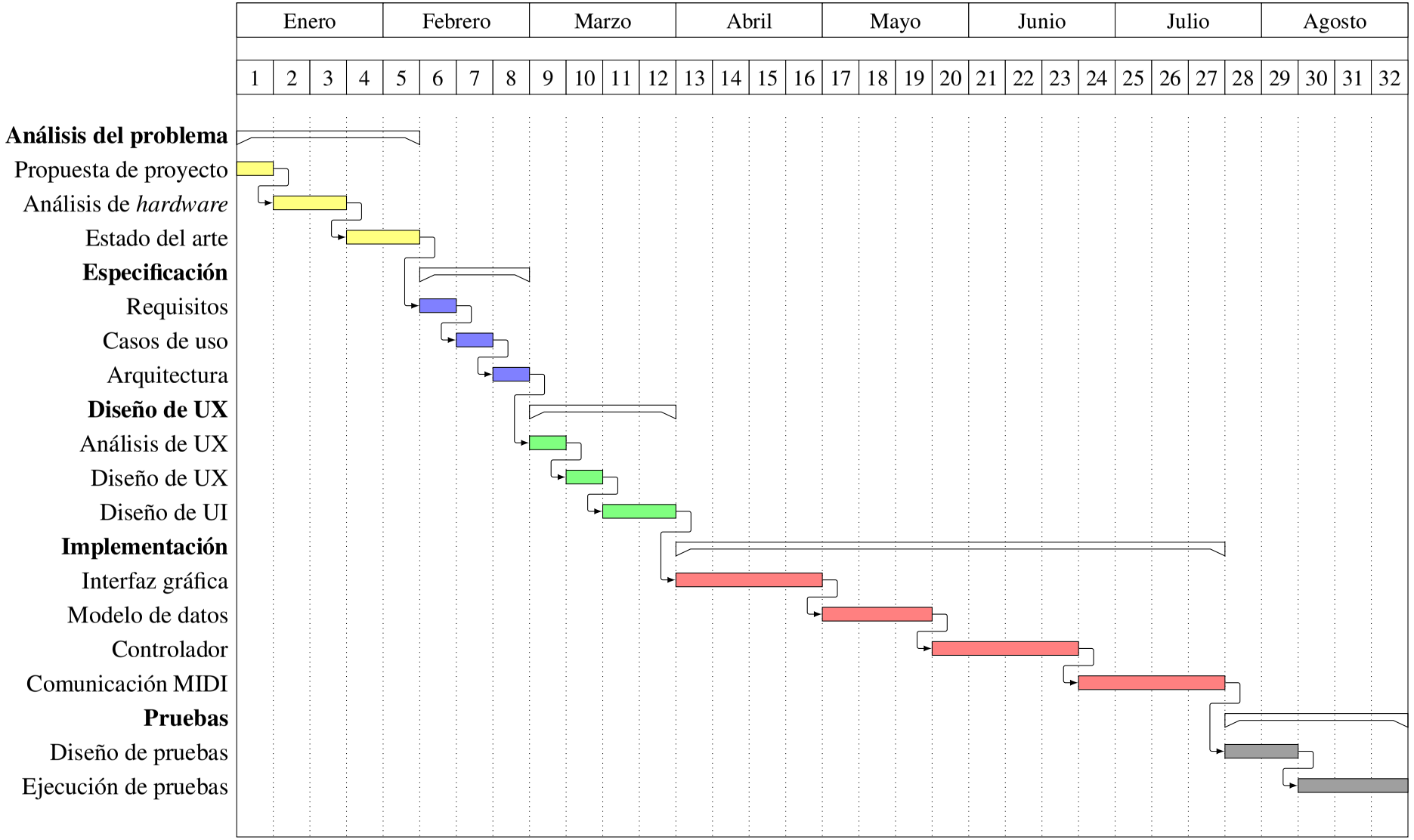


Fig. 11.1. Diagrama de Gantt

12. ANÁLISIS ECONÓMICO

En esta sección se desglosa la estimación del presupuesto del proyecto, así como un análisis y previsión de posibles riesgos y beneficios en el futuro.

12.1. Coste del proyecto

El coste global del proyecto se puede dividir en coste de personal, coste de material y costes indirectos.

12.1.1. Coste de personal

El proyecto ha sido desarrollado por una única persona, pero ya que se han tenido que tratar cuestiones de diseño de UX e ingeniería de *software* aparte de la programación en Java, se puede asumir que su rol ha sido el de desarrollador Java senior, con un sueldo neto medio de 35.983 € en España [44] para una jornada laboral de 40 horas semanales (1.920 horas anuales).

Si se asume un IRPF de 23,78 % se obtiene un sueldo bruto de 51.749 € al año [45], lo que incurre un coste de 26 € la hora. Siguiendo el número de horas totales de la planificación vista en el capítulo 11, 640 horas, se obtiene un coste de personal de 16.640 €.

12.1.2. Coste de material

En este apartado se describen los costes de material, tanto *hardware* como *software*, necesarios para el desarrollo del proyecto, así como costes mínimos en material fungible.

Costes de *hardware*

Los costes de *hardware* se describen en la tabla 12.1 teniendo en cuenta la vida útil del material respecto a la duración del desarrollo del proyecto (8 meses). Se asume que la vida útil del equipo informático es de 5 años [46], y que sólo se indica una unidad de

Producto	Modelo	Precio	Vida útil (meses)	Coste final
Ordenador	PCCom StartUp Pro	858,65 €	60	114,49 €
Monitor	LG 24M35	139 €	60	18,53 €
Ratón	Logitech Trackman	55,43 €	60	7,39 €
Teclado	Redragon K552	53,49 €	60	7,13 €
Controlador MIDI	Electrix Tweaker	179,55 €	60	23,94 €
Total		1286,12 €		**171,48 €**

TABLA 12.1. DETALLE DE LOS COSTES DE *HARDWARE*

Categoría	Producto	Coste final
Sistema operativo	Windows 10 Education	0 €
Entorno de desarrollo integrado	IntelliJ IDEA Community	0 €
Kit de desarrollo de Java	OpenJDK	0 €
Kit de desarrollo de JavaFX	JavaFX SDK	0 €
Bibliotecas de código	Gson	0 €
Sistema de control de versiones	Git	0 €
Plataforma de repositorios	GitHub	0 €
Total		**0 €**

TABLA 12.2. DETALLE DE LOS COSTES DE *SOFTWARE*

cada producto.

Costes de *software*

En el caso del *software*, ya que todas las herramientas utilizadas son o bien *software* libre gratuito o disponen de licencias de estudiante, su coste es 0 €. En la tabla 12.2 se detallan los productos utilizados, que tendrían un coste distinto en el caso de que el proyecto se desarrollara en un ámbito no educativo.

Producto	Cantidad	Coste unitario	Coste final
Paquete de hojas DIN A4	1	5 €	5 €
Bolígrafo BIC Cristal Original Fine negro	4	0,40 €	1,60 €
Bolígrafo BIC Cristal Original Fine azul	2	0,40 €	0,80 €
Post-It pack de 6	1	9,90 €	9,90 €
Consumo energético			19,30 €
Total			**36,60 €**

TABLA 12.3. DETALLE DE LOS COSTES DE MATERIAL FUNGIBLE

Costes de material fungible

Primero se calcula el coste del consumo eléctrico derivado del uso durante 640 horas del equipo informático utilizado para desarrollar el proyecto. Asumiendo un consumo de 201 vatios [47] y un precio de la electricidad de 0,15 € por kilovatio hora, se cuenta con un consumo acumulado de 128.640 vatios o 128,64 kilovatios, por lo que el coste es de 19,30 €.

Todos los costes de material fungible se detallan en la tabla 12.3.

12.1.3. Costes indirectos

Incluyen costes asociados a reuniones, desplazamientos, lugar de trabajo y dietas. Como costes indirectos se puede asumir un 15 % adicional sobre el subtotal de los costes directos.

12.1.4. Coste total

Finalmente, se presenta el coste total del desarrollo del proyecto, incluyendo los subtotales anteriores, en la tabla 12.4. El coste total del proyecto es de 19.375,29 €.

Elemento	Subtotal
Coste de personal	16.640 €
Coste de *hardware*	171,48 €
Coste de *software*	0 €
Coste de material fungible	36,60 €
Subtotal	16.848,08 €
Costes indirectos (15 %)	2.527,21 €
Total	**19.375,29 €**

TABLA 12.4. DETALLE DEL COSTE TOTAL DEL PROYECTO

12.2. Beneficios

No se proyectan beneficios directos a corto o medio plazo ya que el desarrollo del proyecto está pensado para distribuirse de forma gratuita, bajo la licencia de código libre GPL-3. Se plantean posibles beneficios indirectos a corto plazo como donaciones y contribuciones al proyecto, o a medio y largo plazo como ofertas de empleo derivadas de este desarrollo.

12.2.1. Valoración económica del proyecto

Se puede realizar una valoración económica de forma realista ya que este desarrollo compite de forma directa con el editor oficial del Electrix Tweaker, que habrá sido diseñado y desarrollado por una consultoría de *software* o por el equipo interno de Livid o Electrix.

Se asume que, para el desarrollo del editor original, se han dado estas condiciones:

- El equipo consta de tres personas: programador, diseñador de UX y UI, e ingeniero de *software*, con sueldos brutos mensuales de 2100 €, 1850 € y 2850 €, respectivamente [44], con jornadas laborales de 40 horas semanales.
- El trabajo se ha llevado a cabo en 6 meses en total, pero por la disposición y orden de las tareas y roles de cada persona, cada una ha trabajado durante 4 meses.
- Los costes de *hardware* son 600 € mientras que los de *software* ascienden a 900 €.

Elemento	Subtotal
Coste de personal	27.200 €
Coste de *hardware*	600 €
Coste de *software*	900 €
Subtotal	28.700 €
Costes indirectos (15 %)	4.305 €
Subtotal	33.005 €
Margen de beneficio (15 %)	4.950,75 €
Total	**37.955,75 €**

TABLA 12.5. DETALLE DEL VALOR ECONÓMICO APROXIMADO DEL PROYECTO

- Los costes indirectos son de nuevo un 15 % del subtotal de los directos. Los costes de material fungible son despreciables.
- El margen de beneficios es de un 15 % [48], mínimamente por encima de la media para consultorías de *software* pequeñas al tratarse de *software* muy especializado.

Con estas condiciones simplificadas, se puede hacer una estimación del valor económico del proyecto en 37.955,75 €, detallada en la tabla 12.5.

Como se puede observar, la valoración económica es el doble del coste del desarrollo del proyecto presentado en este documento. Esto se refleja en el resultado del proyecto, que requiere un trabajo añadido en áreas como el diseño de interfaz de usuario, y pruebas y distribución para alcanzar el nivel de completitud que tiene la aplicación original.

12.2.2. Riesgos

El mayor riesgo involucrado con el desarrollo de este proyecto es su concentrado impacto socioeconómico y, especialmente, su reducido margen de beneficios. Al competir con aplicaciones disponibles gratuitamente, es difícil tanto éticamente como a nivel de mercado el monetizar este desarrollo. Se trata, en gran parte, de un proyecto realizado por y para la comunidad.

Por otro lado, estas mismas condiciones permiten evitar riesgos de plagio, copia o

ingeniería inversa, ya que ni la idea ni la técnica empleada son innovadoras o exclusivas a este desarrollo.

13. CONCLUSIONES

A continuación se presentan las conclusiones del trabajo, así como las posibilidades de ampliación y trabajo sobre el proyecto en el futuro.

13.1. Conclusiones

En este proyecto se ha desarrollado, primero de manera teórica, una aplicación que funciona como editor del controlador MIDI Electrix Tweaker. Posteriormente, la funcionalidad y diseño teorizados se han implementado en forma de aplicación Java, con JavaFX para generar la interfaz gráfica, haciendo uso del protocolo MIDI para la comunicación con el Tweaker.

Se ha publicado un prototipo en un repositorio público de GitHub con el siguiente enlace: https://github.com/Jorgeelalto/open-tweaker-editor.

A nivel de desarrollo, se han empleado diversas metodologías y técnicas estudiadas en el grado que han permitido definir la estructura y alcance del proyecto, así como su verificación posterior:

- Diseño de *software*, incluyendo la definición de requisitos y casos de uso, el uso del modelo 4+1 para definir la arquitectura del sistema, y los patrones *singleton* y Modelo-Vista-Controlador.
- Diseño y prototipado de la interfaz gráfica siguiendo las heurísticas de Nielsen, el diseño ético y la metodología iterativa, así como las pruebas de usabilidad que permiten estudiar el prototipo desde el punto de vista del usuario final.
- Diseño de pruebas funcionales, y relación de casos de uso, pruebas y requisitos usando matrices de trazabilidad.
- Uso de programación orientada a objetos para plasmar el diseño teórico del *software* en un prototipo funcional de la aplicación.

Utilizando estas técnicas se ha conseguido llegar a un diseño funcional y ampliable en el futuro, y generar un prototipo en un tiempo razonable que sirve como prueba de concepto y como primera versión del producto.

El mayor obstáculo en el desarrollo de este proyecto ha sido el análisis del manual de usuario del Tweaker para entender y aplicar la especificación de mensajes MIDI y SysEx que requiere el controlador para configurar sus parámetros. El manual, a pesar de estar bien escrito y ser suficientemente claro en general, sufre en ocasiones de falta de explicación de terminología o funcionalidades específicas. A pesar de ello, su análisis no ha sido muy costoso y se ha podido implementar gran parte de la funcionalidad descrita.

Por último, a nivel de músico electrónico, la aplicación vuelve a poner en funcionamiento al Electrix Tweaker con todas sus funcionalidades y permitirá que los usuarios puedan volver a usarlo aprovechando todo su potencial. Además, como la aplicación se distribuye de forma gratuita, es compatible con todas las plataformas, y cualquier persona puede hacer los cambios que considere oportunos, llega a todos sus usuarios potenciales.

13.2. Trabajo futuro

El trabajo desarrollado en este proyecto es sólo una parte del potencial con el que cuenta el diseño planteado. En el futuro, se pueden realizar distintas tareas para mejorar y ampliar la aplicación:

- Cambios en la interfaz gráfica, tales como:
  - Implementar la mejoras de la experiencia de usuario reconocidas a través de las pruebas de usabilidad de la sección 8.3.
  - Realizar un rediseño estético de la interfaz usando la especificación de CSS de JavaFX [49], manteniendo la disposición de las secciones pero estilizando los componentes y elementos para mejorar la usabilidad y estética.
- Cambios en la funcionalidad del programa:
  - Desarrollo de funcionalidades diseñadas pero no implementadas en el prototipo.

    - Desarrollo de nuevas características y funciones para conseguir cubrir todos los casos de configuración propuestos por el manual del Electrix Tweaker.
    - Búsqueda y corrección de errores, que a medida que distintos usuarios hagan uso de la aplicación, se documentarán e investigarán.
- Creación de una comunidad de usuarios y desarrolladores alrededor del proyecto, que asegure la continuidad en el tiempo de la aplicación y permita ampliarla de forma conjunta, además de reportar problemas, y proponer soluciones y mejoras.

Finalmente, el mismo proceso de análisis, diseño e implementación se puede aplicar en el futuro a otros controladores o *hardware* MIDI, posiblemente con mayor alcance social si se escoge trabajar sobre uno con mayor cuota de mercado, pudiendo tener un impacto socioeconómico mayor y posibles beneficios a corto y medio plazo.

14. EXTENDED ABSTRACT

14.1. Abstract

This project lays out the design and development of an editing software for the Electrix Tweaker MIDI controller. The application allows the user to configure its parameters easily via a graphical interface, and the communication between the editor and the hardware is managed using using the MIDI protocol through USB.

The design of the application is carried out theoretically by attacking two fronts: on the one hand, the software engineering side, to define the inner workings of the program; on the other, the user experience side, which determines the structure of the graphical interface and how the user will interact with it.

The aim of the application is to be useful to Electrix Tweaker users, replacing the official editor with an open source licensed one that is maintainable and upgradable over time by the community.

14.2. Introduction

There is no doubt that computing has now been introduced in all areas of society, and computers can be found in virtually every workplace around the world. This movement began in the late 1980s, when the first personal computers were introduced [1] making it possible to perform with a single device many of the tasks for which several tools were previously needed. For example, in accounting, ledgers and manual calculations could be replaced by one machine, making the work much faster and facilitating the storage and management of information [2].

This process, however, did not affect everyone equally and the transformation did not take place at the same speed. In the world of audiovisual production, the processing capacity requirements were higher, which made it impossible to replace all the tools used for these jobs with a computer: less CPU-intensive tasks, such as sequencing MIDI or editing digital audio offline (not in real time) have been possible since the late 1990s,

while an application that could fully replace a music studio did not exist until the 2000s [3].

On the other hand, although replacing much of the hardware with virtual instruments and effects made the work faster and simplified many tasks, it was sometimes counterproductive because of the change in the user experience and the way of interacting with them. This is evident when talking about musical instruments or control surfaces: If you replace a set of dedicated physical controls (that allow you to record movements in real time in a comfortable and natural way) with a keyboard and a mouse, you increase the level of flexibility (the user can do more things with only those devices) but the naturalness of the interaction with the system decreases [4].

To replace this user interaction MIDI controllers are used. These are peripherals that allow, through physical controls such as wheels and buttons, to modify the parameters of virtual instruments and effects on the computer. They create a bridge between the usability of hardware and the flexibility of software that brings great value to the audiovisual production workflow in both studio and live situations.

The Electrix Tweaker (figure 14.1) is a MIDI controller released in 2012 in a collaboration between Livid, a manufacturer of highly flexible MIDI control surfaces, and Mixware, the owners of the Electrix name, a digital effects brand [5]. Within the market for this type of hardware, the Tweaker is characterized by having a variety of different controls in a layout geared towards DJing (selecting and playing pre-recorded music to be listened to by an audience) and controllerism (using MIDI controllers to generate live audiovisual content), and by being fully configurable: although it is designed for a certain type of workflow, it lends itself to adapt to any MIDI task.

14.3. Motivation

The Electrix Tweaker has a wide variety of different configuration parameters. Accessing and modifying them through an editor software that is able to display them intuitively using a graphical interface is essential for its effectiveness: without the editor, much of the functionality and usability of the hardware is lost. In this case, the Tweaker has an official editor, shown in figure 14.2, which allows to:

Fig. 14.1. Global view of the Electrix Tweaker

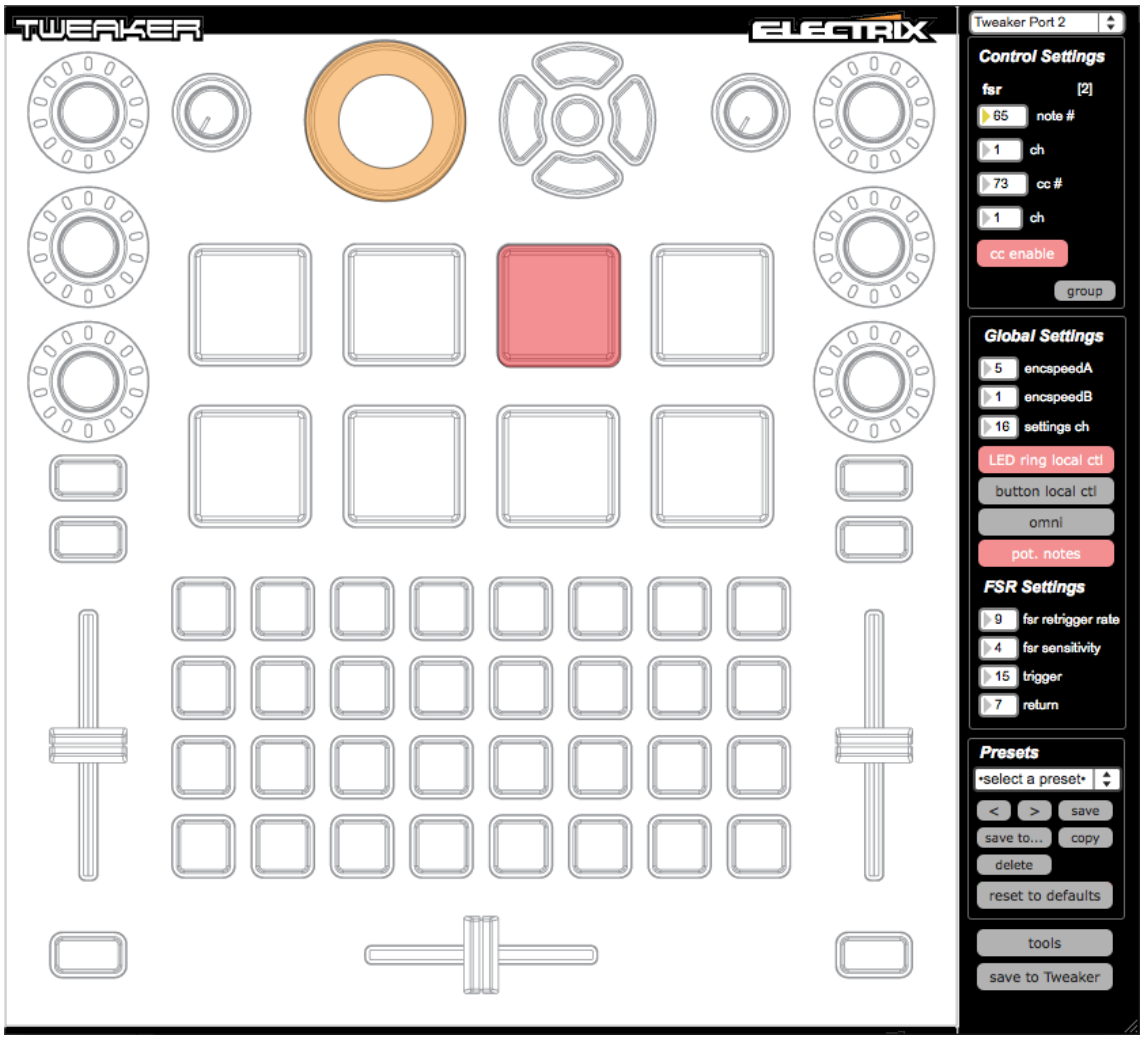


Fig. 14.2. Original editor application

- Modify what MIDI message or messages each control sends.
- Change the colors of the LED lights and their working modes.
- Save the configuration into a local file.
- Load a configuration from a local file and apply it to the hardware.
- Modify the global parameters and establish the current parameters as the default ones.

The official editor is distributed free of charge on the official Electrix website, and was included on a CD packaged with the hardware in the box, in its 1.0 version. During the lifetime of the controller the editor has been updated only once, to version 1.0.1, possibly in October 2014 [6].

The problem that often occurs, including the present case, is that the life expectancy of synthesizers, controllers and hardware modules is generally longer than that of software editors and virtual instruments. While the operation of an editor depends on the computer being able to physically run the code and the operating system being compatible with it, a MIDI controller as generic as the Tweaker can function until the materials and components themselves degrade. In general, synthesizers and MIDI devices, if solidly built and properly maintained, can last for decades [7], as they are generally intended for heavy use in live situations, where they are exposed to the most wear and tear.

In this case, the Electrix Tweaker Editor was last updated in 2014, and thus suffers from several compatibility issues with 2021 operating systems:

- In macOS 11 the Editor is not certified, so if you try to run it the operating system blocks the operation. You can temporarily avoid this behavior by changing the security settings, but it is a suboptimal workaround if you need the editor to be easily accessible.
- In Windows 10 the Editor runs but does not recognize the driver even though it is properly connected. The operating system does recognize the MIDI device, and so it works with other programs (Reaper, FL Studio and Mixxx).
- On Linux the Editor does not work. There are no versions for any distribution of this operating system.

A possible solution to this problem is to release the source code of the Editor. Free software is software whose users have the ability to run, distribute and modify its code, among other freedoms [8]. In the case of an editor for a MIDI controller that is distributed free of charge and whose functionality is fully documented, if the original developer can no longer support the application there is no compelling reason not to release the original code with a license that allows users to at least solve compatibility problems. In the case of the original Tweaker Editor, the code could be analyzed and debugged to fix the bug that prevents it from recognizing the driver.

14.4. Objectives

The objective of this work is, on the one hand, to solve the problems raised in section 14.3, by developing an application that is:

- Useful for users of Electrix Tweaker, i.e. to function as a substitute for the official editor, based on it and allowing to edit at least the most used settings.
- Free software, so that the community can continue to support the application if necessary without all the maintenance responsibility falling on a few people or a single entity, and so the effort to implement all the features offered by the original editor is potentially collaborative.
- Cross-platform. Since the driver is independent of the operating system to which it connects, it makes sense that the editor also works on at least the three main operating systems (Windows, macOS, and Linux).
- Easy to develop and stable, to get at least a prototype of the application that works properly but is also extensible in the long term.

On the other hand, it is the exploration of current software solutions that allow the design and programming of a GUI application with capabilities to send and receive information via the MIDI protocol. Given that it has limited use cases and that the demand for this type of application is low, the available options are fewer, allowing for a quick analysis.

14.5. Proposed system

The proposed solution is the design and development of a Java application to manage the editing of controller parameters, in which the user can use a graphical interface to select each control, edit its parameter values and send them to the Electrix Tweaker at will.

After the analysis carried out in section 2, those technologies were chosen because they check all the requirements of the project, as seen on section 14.4. The use of this programming language facilitates the operation of the final result in the three operating

systems previously seen, and also incorporates libraries to manage communications with the MIDI protocol to send messages to the hardware by default. On the other hand, JavaFX will be used for the design and implementation of the graphical interface.

The software solution is designed on sections 4.1, 5.2 and 6. The user interface and UX of the system is developed and explained in section 7.

14.5.1. Testing

For this project, unit tests and usability tests have been designed.

The purpose of unit tests is to verify the correct operation of the internal components of the internal components of the application individually. In the case of this project we have chosen to develop this type of tests for the TweakerConfig component, the data model, since it is the only component with enough methods that do not interact with the hardware. data model, since it is the only component with enough methods that do not interact with the hardware or the graphical interface.

In the case of usability tests, these aim to verify whether the user experience of the system is adequate by means of real tests with potential users. For the purpose of this project, usability test cases are designed based on the use cases defined in section defined in section 8.2.1.

These usability tests have been performed with 6 test subjects with varying degrees of engineering, musical and MIDI understanding. The results, as seen in section 8.3.1, show that several changes in the design of user experience are needed for this application to cater all potential users.

14.6. Conclusions

In this project an application that works as an editor of the MIDI controller Electrix Tweaker has been developed, first in a theoretical way. Afterwards, the theorized functionality and design have been implemented in the form of a Java application, with JavaFX to generate the graphical interface, making use of the MIDI protocol to communicate with the Tweaker.

A prototype of the application has been published as a public GitHub repository with the next link: https://github.com/Jorgeelalto/open-tweaker-editor.

At the development level, several methodologies and techniques studied in the degree have been used to define the structure and scope of the project, as well as its subsequent verification:

- Software design, including the definition of requirements and use cases, the use of the 4+1 model to define the system architecture, and the singleton and Model-View-Controller patterns.
- Design and prototyping of the graphic interface following the Nielsen heuristics, ethical design and iterative methodology, as well as usability tests that allow studying the prototype from the end user's point of view.
- Design of functional tests and relationship of use cases, tests and requirements using traceability matrices.
- Use of object-oriented programming to translate the theoretical design of the software into a functional prototype of the application.

By using these techniques, it has been possible to arrive at a functional and scalable design, and to generate a prototype in a reasonable time that serves as a proof of concept and as the first version of the product.

The biggest obstacle in the development of this project has been the analysis of the Tweaker user manual to understand and apply the specification of MIDI and SysEx messages required by the controller to configure its parameters. The manual, although well written and clear enough in general, sometimes suffers from lack of explanation of terminology or specific functionalities. In spite of this, it has not been very costly to analyze and it has been possible to implement much of the functionality described.

Finally, at the electronic musician level, the application brings the Electrix Tweaker back into operation with all its functionalities and will allow users to use it to its full potential again. Moreover, it reaches all its potential users since the application is distributed free of charge, is compatible with all platforms, and anyone can make changes as they see fit.

BIBLIOGRAFÍA

[1] *Personal Computer History: 1975-1984 | Low End Mac*, https://lowendmac.com/2014/personal-computer-history-the-first-25-years/, (acceso: 22/07/2021).

[2] *How Modern Technology Has Changed Accounting | Cheryl Jefferson & Associates*, https://www.cjeffersoncpa.com/qb/how-modern-technology-has-changed-accounting/, (acceso: 22/07/2021).

[3] *Recording Basics: The History of the DAW*, https://hub.yamaha.com/proaudio/pa-history/the-history-of-the-daw/, (acceso: 23/07/2021).

[4] *Software Mixing With Hardware MIDI Controllers*, https://www.soundonsound.com/techniques/software-mixing-hardware-midi-controllers, (acceso: 23/07/2021).

[5] *Livid Instruments*, http://lividinstruments.com/news/electrix-tweaker/, (acceso: 23/07/2021).

[6] *Downloads – Electrix*, https://www.electrixpro.com/downloads/, (acceso: 20/07/2021).

[7] *Restoring Analogue Synthesizers*, https://www.soundonsound.com/techniques/restoring-analogue-synthesizers, (acceso: 20/07/2021).

[8] *What is Free Software? - GNU Project - Free Software Foundation*, https://www.gnu.org/philosophy/free-sw.en.html, (acceso: 20/07/2021).

[9] *How Rotary Encoders Work - Electronics Basics - The Geek Pub*, https://www.thegeekpub.com/245407/how-rotary-encoders-work-electronics-basics/, (acceso: 20/07/2021).

[10] *Summary of MIDI 1.0 Messages*, https://www.midi.org/specifications-old/item/table-1-summary-of-midi-message, (acceso: 20/07/2021).

[11] *What is MIDI? The Essential Guide*, https://www.sweetwater.com/insync/midi-essential-guide/, (acceso: 19/07/2021).

[12] D. M. Huber, *The MIDI Manual*. Sams, 1991.

[13] *MIDI tutorial for programmers*, https://www.cs.cmu.edu/~music/cmsip/readings/MIDI%20tutorial%20for%20programmers.html, (acceso: 23/07/2021).

[14] B. C. Florea, "MIDI-based controller of electrical drives," en *Proceedings of the 2014 6th International Conference on Electronics, Computers and Artificial Intelligence (ECAI)*, 2014, pp. 27-30. DOI: 10.1109/ECAI.2014.7090159.

[15] *Basics of USB-MIDI* -, https://www.midi.org/midi-articles/basic-of-usb, (acceso: 19/07/2021).

[16] *Introducing MIDI 2.0*, https://www.soundonsound.com/music-business/introducing-midi-20, (acceso: 23/07/2021).

[17] *The RtMidi Tutorial*, https://www.music.mcgill.ca/~gary/rtmidi/, (acceso: 26/07/2021).

[18] *python-rtmidi · PyPI*, https://pypi.org/project/python-rtmidi/, (acceso: 26/07/2021).

[19] *rtmididrv · pkg.go.dev*, https://pkg.go.dev/gitlab.com/gomidi/rtmididrv, (acceso: 26/07/2021).

[20] *Java Sound API*, https://www.oracle.com/java/technologies/java-sound-api.html, (acceso: 26/07/2021).

[21] *Web MIDI API*, https://webaudio.github.io/web-midi-api/, (acceso: 26/07/2021).

[22] *MIDIAccess - Web APIs | MDN*, https://developer.mozilla.org/en-US/docs/Web/API/MIDIAccess, (acceso: 26/07/2021).

[23] *The Qt Company*, https://www.qt.io/company, (acceso: 24/07/2021).

[24] *How do I configure Qt for cross-compilation from Linux to Windows target? - Stack Overflow*, https://stackoverflow.com/questions/10934683/how-do-i-configure-qt-for-cross-compilation-from-linux-to-windows-target, (acceso: 24/07/2021).

[25] *Electron considered harmful*, https://drewdevault.com/2016/11/24/Electron-considered-harmful.html, (acceso: 24/07/2021).

[26] *JavaFX history - JavaFX Essentials*, https://subscription.packtpub.com/book/web_development/9781784398026/1/ch01lvl1sec10/javafx-history, (acceso: 24/07/2021).

[27] *Frequently Asked Questions (FAQ) - The Go Programming Language*, https://golang.org/doc/faq#Origins, (acceso: 25/07/2021).

[28] *Cross Compiling*, https://golangcookbook.com/chapters/running/cross-compiling/, (acceso: 25/07/2021).

[29] *GUI | Learn Go Programming*, https://golangr.com/gui/, (acceso: 25/07/2021).

[30] *Graphic User Interface FAQ — Python 3.9.6 documentation*, https://docs.python.org/3/faq/gui.html, (acceso: 25/07/2021).

[31] *Ctrlr – Control your MIDI life (MIDI editor for all your hardware)*, https://ctrlr.org/, (acceso: 31/07/2021).

[32] *4 + 1 Architectural View Model: Introduction · Dev Cycles*, https://devcycles.io/2019/02/27/4-1-architectural-view-model-introduction/, (acceso: 16/08/2021).

[33] *What is Object Oriented Programming? OOP Explained in Depth*, https://www.educative.io/blog/object-oriented-programming, (acceso: 16/08/2021).

[34] *The Model View Controller Pattern – MVC Architecture and Frameworks Explained*, https://www.freecodecamp.org/news/the-model-view-controller-pattern-mvc-architecture-and-frameworks-explained/, (acceso: 18/08/2021).

[35] *dartellisonfeilerhabermann.pdf*, https://www.ics.uci.edu/~andre/ics228s2006/dartellisonfeilerhabermann.pdf, (acceso: 19/08/2021).

[36] *Ethical UX-Design Guidelines. As the UX-Professional on the team, you... | by uxEngineering | Aug, 2021 | UX Planet*, https://uxplanet.org/ethical-ux-design-guidelines-26f0767ed085, (acceso: 19/08/2021).

[37] *10 Usability Heuristics for User Interface Design*, https://www.nngroup.com/articles/ten-usability-heuristics/, (acceso: 03/09/2021).

[38] *The jpackage Command*, https://docs.oracle.com/en/java/javase/14/docs/specs/man/jpackage.html, (acceso: 29/08/2021).

[39] *Packaging Overview*, https://docs.oracle.com/en/java/javase/14/jpackage/packaging-overview.html, (acceso: 29/08/2021).

[40] *GNU General Public License v3 (GPL-3) Explained in Plain English - TLDRLegal*, https://tldrlegal.com/license/gnu-general-public-license-v3-(gpl-3), (acceso: 20/08/2021).

[41] *Google Java Style Guide*, https://google.github.io/styleguide/javaguide.html, (acceso: 20/08/2021).

[42] *Overview of the MIDI Package (The Java™ Tutorials >Sound)*, https://docs.oracle.com/javase/tutorial/sound/overview-MIDI.html, (acceso: 20/08/2021).

[43] *the_2016_namm_global_report.pdf*, https://www.quilterlabs.com/images/quilterimg/uiimages/752/the_2016_namm_global_report.pdf, (acceso: 25/08/2021).

[44] *Sueldo: Senior Java Developer | Glassdoor*, https://www.glassdoor.es/Sueldos/senior-java-developer-sueldo-SRCH_KO0,21.htm?clickSource=searchBtn, (acceso: 22/08/2021).

[45] *BBVA*, https://web.bbva.es/public.html#public/calculadora-sueldo-neto, (acceso: 22/08/2021).

[46] *What Is the Life Span of the Average PC?* https://smallbusiness.chron.com/life-span-average-pc-69823.html, (acceso: 22/08/2021).

[47] *Power Supply Calculator - PSU Calculator | OuterVision*, https://outervision.com/power-supply-calculator, (acceso: 22/08/2021).

[48] *The profitability of software development firms varies by size*, https://greatnotbig.com/2011/06/profitability-varies-by-firm-size/, (acceso: 23/08/2021).

[49] *JavaFX CSS Reference Guide*, https://openjfx.io/javadoc/16/javafx.graphics/javafx/scene/doc-files/cssref.html, (acceso: 04/09/2021).