Escola Tècnica Superior d'Enginyeria Informàtica
Universitat Politècnica de València

Automatización de la salida de controladores MIDI no motorizados

Trabajo Fin de Grado

Grado en Ingeniería Informática

Autor: Jaime Manuel Polo Esteve

Tutor: Luís José Saiz Adalid

CURSO ACADÉMICO 2014-2015

Resumen

Los controladores MIDI permiten la gestión hardware del software musical. Como un teclado MIDI permite la entrada de notas a un secuenciador MIDI, cuando se carga un proyecto en el secuenciador o cuando se cambian valores gestionados por el controlador, como no se pueden mover si no son motorizados, se pierde el valor. La solución que se propone en este proyecto consiste en intercalar un programa entre el controlador y el secuenciador, que memorice los valores y permita una gestión posterior más inteligente del controlador MIDI.

La solución que propone este proyecto sirve para poder utilizar controladores MIDI con controles movibles sin estar motorizados. Esta propuesta se basa en un programa que interpreta los valores anteriores junto con los valores aportados por la entrada del controlador para ofrecer un valor apropiado.

Los valores apropiados son calculados según los distintos algoritmos propuestos en la solución:

Uno de estos algoritmos permite recoger el valor de entrada independientemente del valor anterior y mostrarlo en el secuenciador.

Otro algoritmo sirve para hacer que el valor sea buscado con el controlador hasta que sea encontrado y así sincronizar los valores de entrada y salida.

Por último, el más importante de los algoritmos, uno que permite que el valor del controlador en el secuenciador sea proporcional al valor anterior que tiene el secuenciador teniendo en cuenta la posición inicial que tiene el controlador movible en el controlador MIDI de entrada.

Palabras clave: Controlador MIDI, secuenciador MIDI, hardware musical, software musical.

Resum

Els controladors MIDI permeten la gestió del programari musical per mitjà del maquinari. Com un teclat MIDI permet l'entrada de notes a un seqüenciador MIDI, quan es carrega un projecte en el seqüenciador o quan es canvien valors gestionats pel controlador, com no es poden moure si no són motoritzats, es perd el valor. La solució que es proposa en este projecte consistix a intercalar un programa entre el controlador i el seqüenciador, que memoritze els valors i permeta una gestió posterior més intel·ligent del controlador MIDI.

La solució que proposa este projecte servix per a poder utilitzar controladors MIDI amb controls movibles sense estar motoritzats. Esta proposta es basa en un programa que interpreta els valors anteriors junt amb els valors aportats per l'entrada del controlador per a oferir un valor apropiat.

Els valors apropiats són calculats segons els distints algoritmes proposats en la solució:

Un d'estos algoritmes permet arreplegar el valor d'entrada independentment del valor anterior i mostrar-ho en el seqüenciador.

Un altre algoritme servix per a fer que el valor siga buscat amb el controlador fins que siga trobat i així sincronitzar els valors d'entrada i eixida.

Finalment, el més important dels algoritmes, un que permet que el valor del controlador en el seqüenciador siga proporcional al valor anterior que té el seqüenciador tenint en compte la posició del controlador movible en el controlador MIDI d'entrada.

Paraules clau: Controlador MIDI, seqüenciador MIDI, maquinari musical, programari musical.

Abstract

The MIDI hardware controllers allow music software management. As a MIDI keyboard allows entry of notes to a MIDI sequencer, when loading a project in the sequencer or when values managed by the controller are changed, and they cannot be moved if they are not motorized, the value is lost values are changed. The solution proposed in this project is to collate a program between the controller and the sequencer, to memorize the values and allow a more intelligent MIDI controller subsequent management.

The solution proposed by this project serves to use MIDI controllers with movable non-motorized controls. This proposal is based on a program that interprets the above values with the values provided by the controller input to provide the appropriate value.

The appropriate values are calculated for different algorithms proposed in the solution:

One of these algorithms allows collecting input value regardless of the previous value and displaying it in the sequencer.

Another algorithm is used to set the value to be searched with the controller until it is found and thus synchronize the input and output values .

Finally, the most important algorithm, one that allows the controller value in the sequencer being proportional to the previous value of the sequencer considering the initial position of the movable controller in the MIDI input controller.

Keywords: MIDI Controller, MIDI Sequencer, Music hardware, Music software.

Tabla de contenidos

1 Introducción 13
1.1 Motivación 13
1.2 Objetivos 16
2 Estado actual 17
2.1 Alternativas existentes: ventajas y deficiencias 19
2.2 Conceptos teóricos 20
2.2.1 Sonido 20
2.2.2 Conversores A/D y D/A 20
2.2.3 Música 21
2.2.4 Sintetizadores 21
2.2.5 Sampler 22
2.2.6 Controlador MIDI 23
2.2.7 Música por ordenador. Tarjetas de sonido. 25
2.2.8 Virtualización de dispositivos audio y MIDI 25
3 Especificación de requisitos 29
3.1 Requisitos 29
3.2 El Protocolo MIDI 30
3.2.1 Los mensajes MIDI 31
3.3 El soporte multimedia en Windows 37
4 Diseño e implementación 43
4.1 Abstración 44
4.2 Codificación 45
4.2.1 La clase win32 46
4.2.2 La clase Midiio 51
4.3 Ejecución 59
4.4 Vídeo demostrativo 60
5 Conclusiones y trabajo futuro 61
5.1 Conclusiones 61
5.2 Trabajo futuro 61
6 Bibliografía 63

Ilustraciones

Ilustración 1. Caso con controladores no motorizados..... 14
Ilustración 2. Caso con controladores motorizado..... 14
Ilustración 3. Caso del proyecto..... 15
Ilustración 4. Secuenciador, instrumentos y efectos..... 17
Ilustración 5. Mixer 17
Ilustración 6. Nano Kontrol de Korg..... 18
Ilustración 7. Launch Control de Novation 18
Ilustración 8. MPD18 de Akai profesional..... 19
Ilustración 9. Sintetizador Roland TB-303..... 21
Ilustración 10. Sampler modular para rack. Akai profesional. 22
Ilustración 11. Conectores MIDI. 23
Ilustración 12. Teclado maestro MIDI Behringer UMX25. 24
Ilustración 13. Sistema audio/midi virtual. 27
Ilustración 14. Conexión MIDI entre teclado y sintetizador. 30
Ilustración 15. MIDI thru 31
Ilustración 16. Canales MIDI. 32
Ilustración 17. Pistas MIDI.....32
Ilustración 18. Mensaje MIDI. 33
Ilustración 19. Descripción de los mensajes MIDI. 34
Ilustración 20. Especificaciones MIDI del sintetizador Yamaha DX1. 36
Ilustración 21. General MIDI. Program change. 37
Ilustración 22. Obtener capacidades de los dispositivos MIDI. 39
Ilustración 23. Capacidad de los dispositivos MIDI. 40
Ilustración 24. Abrir dispositivos MIDI..... 40
Ilustración 25. Función CallBack. 41
Ilustración 26. Enviar datos MIDI. 41
Ilustración 27. La clase win32. Definición de constantes y estructuras... 47
Ilustración 28. La clase win32. Definición de funciones o métodos. 50
Ilustración 29. Esqueleto MidiInProc.....52
Ilustración 30. Clase Midiio. 58
Ilustración 31. Área de notificaciones de Windows e icono. 59
Ilustración 32. Ventana del programa MIDIm. 59

1 Introducción

Actualmente la informática musical ha avanzado hasta el punto de que la entrada de datos MIDI, notas y controles, se realiza mediante los controladores MIDI. De la misma forma que un teclado MIDI permite enviar las notas a un secuenciador o sintetizador que las almacene o las interprete, en cada caso, un controlador MIDI puede enviar valores que pueden ser utilizados para variar otros parámetros, más de funcionamiento y control como podrían ser el volumen, el panorama, los valores de efectos, etc.

1.1 Motivación

Los controladores MIDI hardware (en adelante solo controladores MIDI) con potenciómetros no motorizados no permiten recibir datos y moverse de forma automática a la posición definida en el secuenciador para un valor de control.

Vamos a definir los agentes que normalmente intervienen:

- El controlador MIDI, permite establecer valores mediante controles. Normalmente, y como se verá después, los controladores MIDI suelen tener valores comprendidos entre 0 y 127. Estos valores corresponden con el valor mínimo y máximo a representar.

- El secuenciador, gestiona los datos MIDI, tanto notas como datos de control en el tiempo. Es capaz de recoger los valores de un controlador MIDI de entrada y representarlo en pantalla.

- El cable MIDI, uno para entrada y otro para salida, entre dispositivos.

Como se representa en la Ilustración 1, los valores que generamos en el controlador MIDI son pasados al secuenciador sin problemas pero cuando cambia algún valor en el secuenciador los valores se pierden dado que dicho tipo de controlador MIDI no tiene entrada de valores o aunque la tuviese no sabría cómo representar dichos valores.

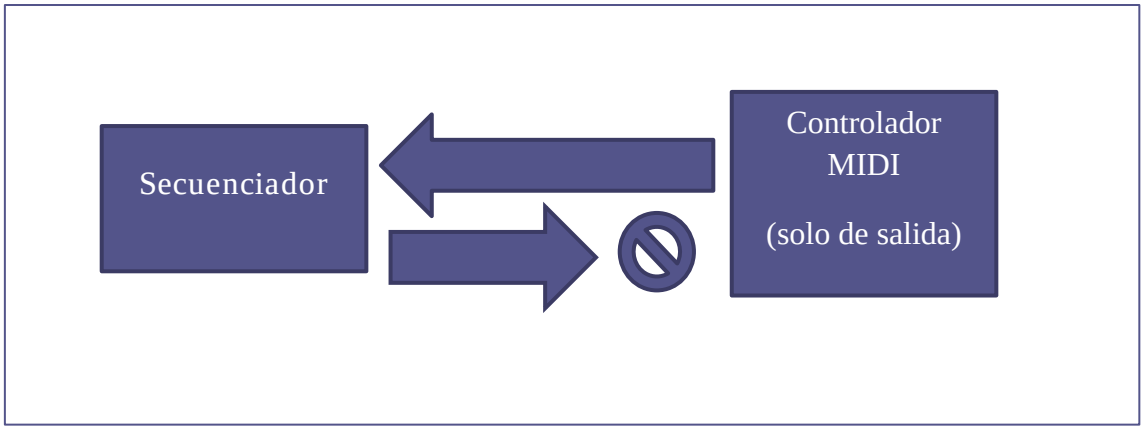


Ilustración 1. Caso con controladores no motorizados.

Por ejemplo, imaginemos que subimos el volumen a la pista 1 con el controlador MIDI. Para ello, movemos el controlador y lo dejamos en la posición, por ejemplo, 64, que es justo la mitad del potenciómetro. Este valor es enviado y adoptado por el secuenciador. Si cambiásemos el valor en el secuenciador y lo colocamos a 10 de volumen: ¿Cómo va a ser recogido el valor por el controlador MIDI si no dispone de entrada ni este es motorizado para representarlo? El valor seguirá en 64 (en el controlador MIDI) tal y como lo habíamos dejado en el momento anterior aunque en el secuenciador esté el valor 10. Además ¿qué pasará si movemos el controlador MIDI al valor 70? Según el modelo de la Ilustración 1, en el secuenciador se producirá un salto del 10 al 70 para representar dicho valor.

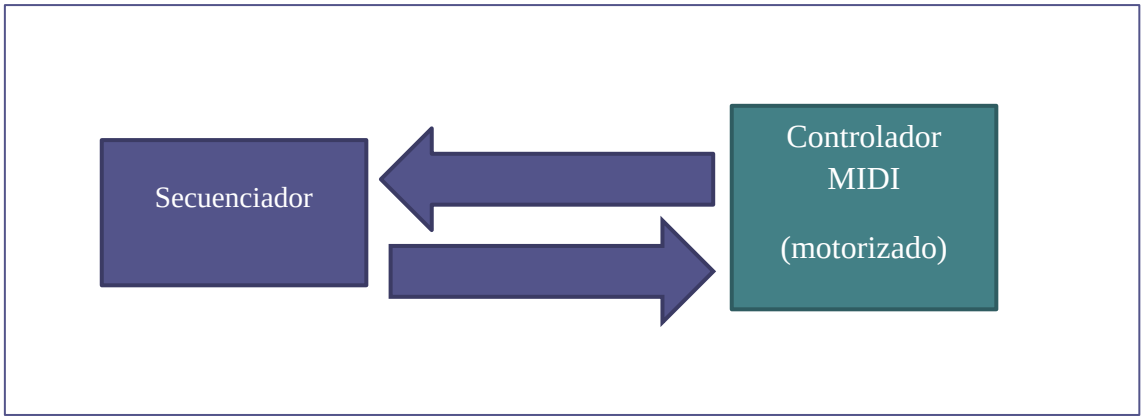


Ilustración 2. Caso con controladores motorizado.

En el segundo caso, tal y como se representa en la Ilustración 2, el secuenciador emite el valor del control y es recogido por el Controlador MIDI que sí que sabe cómo representarlo. Mediante el potenciómetro motorizado se mueve a la posición que representa el valor.

Este tipo de controladores con potenciómetros motorizados tiene la capacidad de representar valores de entrada al controlador. En este caso, el de la Ilustración 2, el motor hace que el controlador se mueva al valor que ha sido establecido en el secuenciador de forma automática. El controlador MIDI con controles motorizados es un dispositivo de entrada/salida: Guarda el valor que le entra y el valor que establecemos nosotros al moverlo como de salida

La solución planteada en la Ilustración 2 es la ideal, solo que este tipo de controladores son menos numerosos en el mercado y sobre todo son mucho más caros.

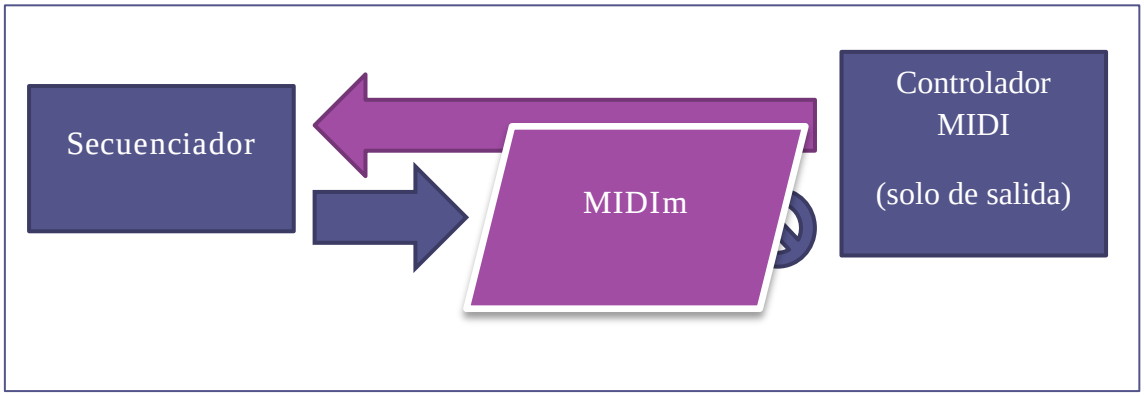


Ilustración 3. Caso del proyecto.

En la Ilustración 3 se plantea la solución propuesta en este proyecto para los casos de la Ilustración 1 y Ilustración 2. Consiste en comunicar dicho controlador MIDI con el software de secuenciador tanto de entrada como de salida con software intermedio que haga de memoria y que interprete los nuevos valores de entrada al secuenciador teniendo en cuenta los valores de salida de este. Sigue sin haber comunicación de entrada al controlador MIDI pero los efectos de esa falta de comunicación se minimizan con el memorizador de valores (MIDIm).

El controlador seguirá sin poder recoger los valores de salida del secuenciador pero estos sí que se guardarán. Se guardan en el programa MIDIm. Y se podrán interpretar cuando se tengan que enviar de vuelta al secuenciador para que se devuelvan de forma que no se envíe nada hasta alcanzar el valor buscado en el controlador MIDI o enviándose de forma suave y proporcional al valor actual del secuenciador.

1.2 Objetivos

El objetivo del proyecto es la realización de un programa que se situará entre un secuenciador y un controlador mediante comunicación MIDI. Por un lado guardará los valores que se cambien en el secuenciador, es decir, servirá de memoria para valores de controles MIDI y por otro lado, reinterpretará los valores que salen del controlador MIDI hacia el secuenciador dependiendo de los valores memorizados anteriormente descritos.

He pensado en realizar varios tipos de interpretación de los valores de entrada al secuenciador en el caso de que se produzcan cambios en los valores de los controladores de éste:

- Tipo normal: Para este caso el sistema funcionará como si no hubiese memorizador y el resultado será un cambio brusco en el secuenciador. Como ocurre en el modelo representado en la Ilustración 1 pero para el uso en la Ilustración 3.

- Tipo buscar: En el que no se enviarán datos al secuenciador hasta que se encuentre el valor que tenía el secuenciador cuando se cambió en él. Ver Ilustración 3.

- Tipo proporcional: En este tipo se calculará proporcionalmente el valor a enviar al secuenciador teniendo en cuenta la diferencia entre el valor del controlador MIDI y el del secuenciador. Ver Ilustración 3.

2 Estado actual

La forma que tenemos de controlar los parámetros de un secuenciador y los módulos o plug-ins de efectos e instrumentos es mediante los controladores MIDI. Estos pueden ser tanto hardware como software, físicos o virtuales [1].

Ilustración 4. Secuenciador, instrumentos y efectos.

Los virtuales necesitan de un sistema informático para funcionar, mientras que los de tipo hardware necesitan tener componentes electrónicos en forma de botones, deslizantes, rotatorios, pads, etc.

Ilustración 5. Mixer

Botones para poder simular conexiones o desconexiones, por ejemplo activar o desactivar pista, o para hacer “mute” o “solo”, etc. Poner la pista en modo “mute” evita en la reproducción de la pista cuando se reproduce el proyecto. En cambio, aplicar “solo” a la pista hace que solo se reproduzca dicha pista; el resto se silencia. En la Ilustración 6 podemos ver un controlador MIDI de Korg: Nano Kontrol[2].

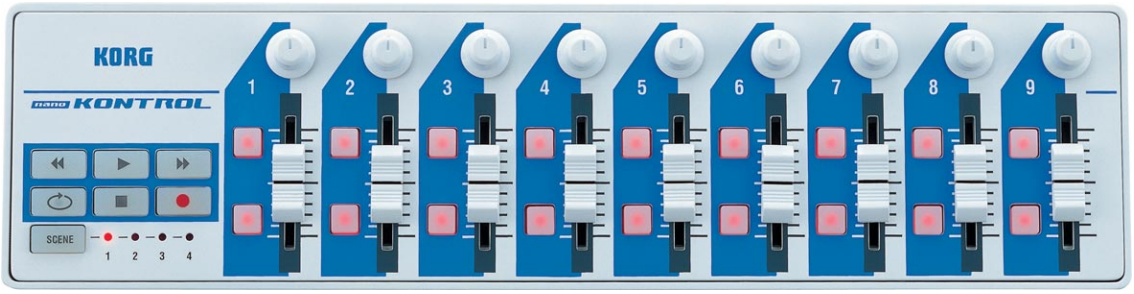


Ilustración 6. Nano Kontrol de Korg

También podemos encontrar deslizantes o rotatorios que comprenden valores en una escala de 0 a 127 para los distintos valores que pueden tener los controladores MIDI. Ver el protocolo MIDI en el apartado 3.2. En la Ilustración 7 podemos ver el launch Control de Novation[3].

Ilustración 7. Launch Control de Novation

También hay pads que permiten la activación de una nota mientras se está pulsando y la desconexión de ésta cuando se deja de pulsar, para representar la ejecución de una nota como en una tecla de piano o percusión. Ver Note On y Note Off en el apartado 3.2.1. En la Ilustración 8 podemos encontrar el MPD18 de Akai[4].

Ilustración 8. MPD18 de Akai profesional.

2.1 Alternativas existentes: ventajas y deficiencias

Hay muchas posibilidades actualmente pero todas ellas aportan ventajas e inconvenientes que siempre hay que tener en cuenta.

Por un lado, tenemos mesas con controladores motorizados que, como he dicho anteriormente, permiten posicionarse interpretando el valor en el punto adecuado. Este tipo de mesas son más caras ya que los controladores son más caros dado sus posibilidades y de los motores de los que están compuestas.

Por otro lado, existen mesas que disponen de controladores rotatorios pero de rotación infinita y que representan el valor mediante algún led situado alrededor del control. El problema es que ese indicador led no es fino, no define un valor claramente, y por lo tanto más bien indica un intervalo de valores.

En el caso que nos ocupa, hay mesas que disponen de controladores con limitación de 0 a 127 (que es lo que se necesita para representar valores MIDI) y su valor representado será el que marquemos en ese momento, el que esté en ese momento o proporcional al que esté en ese momento.

2.2 Conceptos teóricos

2.2.1 Sonido

El sonido es la transmisión de cambios de presión en las moléculas del aire de una fuente que vibra al oído humano y la interpretación de este por parte del cerebro. Según el objeto y la mezcla de la vibración de estos se producen los distintos sonidos. Además esta propagación de las ondas sonoras por el aire produce interacciones con otros objetos. Se producen reflexiones en algunos objetos y absorciones en otros. Y al final la suma de todo esto es lo que oímos e interpretamos como sonido.

2.2.1.1 Sonido analógico

Aquellos sonidos que son almacenados como diferencias de voltaje en donde hay diferencias de presión son analógicos. Es una forma de almacenar el sonido mediante una función continua del voltaje en el tiempo.

2.2.1.2 Sonido digital

Consiste en una representación discreta del sonido (secuencia de valores) en el tiempo. Para ello se toman un número de muestras cada segundo con una resolución b de los distintos canales. Por ejemplo, el sonido digital en un CD musical está almacenado tomando 44100 muestras de 16 bits de tamaño por cada canal: 44100 x 16 x 2 = (aprox.) 172 KBytes por segundo.

2.2.2 Conversores A/D y D/A

Compuestos de componentes electrónicos que permiten la transmisión entre el sonido analógico y digital

2.2.3 Música

La música está formada por los sonidos musicales. Los cuales tienen distintas aspectos físcos del sonido: alturas, timbres e intensidades.

Las alturas están definidas por las escalas musicales que forman las notas y por las octavas. Son los doce semitonos que van desde do a do (do re mi fa sol la si do) con sus semitonos intermedios. La altura tiene que ver con la frecuencia del sonido

El timbre es como el tipo del sonido según la fuente. No es lo mismo el sonido producido por un Do de flauta que un Do de guitarra. Tienen timbre distinto. El timbre es el color del sonido.

La intensidad se mide en decibelios y podríamos decir que es “lo fuerte que está el sonido”. La intensidad se refiere a la amplitud del sonido.

2.2.3.1 Música electrónica

Música generada con dispositivos electrónicos. Véase los sintetizadores.

2.2.4 Sintetizadores

En la música tradicional, los instrumentos generan los distintos tipos de sonidos. Con la llegada de la música electrónica aparecen los sintetizadores. Los sintetizadores son dispositivos electrónicos que mediante elementos sencillos generan los distintos sonidos. Estos elementos sencillos están compuestos normalmente de osciladores electrónicos que combinados pueden incluso imitar a los tradicionales. Se aplican otros elementos para conseguir el envolvente, efectos, filtros, etc.

Ilustración 9. Sintetizador Roland TB-303.

Se utilizan algunas técnicas para generar los sonidos que crean los sintetizadores.

Por ejemplo, la síntesis aditiva, utiliza la técnica de suma de ondas sencillas (sinusoidales) para formar el sonido. Se basa en el Teorema de Fourier.

Con la modulación de frecuencia (FM) también es posible generar sonidos que a veces resultan parecidos a ciertos instrumentos.

La síntesis de tabla de ondas consiste en la digitalización de sonidos reales para almacenarlos y posteriormente generar instrumentos basados en ellos. Estos sonidos son almacenados en la ROM de los dispositivos que disponen de esta técnica. Incluso se toman muestras a distintas alturas e intensidades de instrumentos naturales para luego formarlos en la síntesis del sonido de salida y dar un resultado de imitación excelente.

En la Ilustración 9 podemos encontrar una imagen del famoso sintetizador y línea de bajo, Roland TR-303[5]. Tiene capacidades de secuenciador y permite aplicar filtros y efectos. Incluso existen versiones virtuales de este como el ya obsoleto ReBirth[6].

2.2.5 Sampler

Los “samplers” son dispositivos electrónicos que permiten reproducir sonidos. Estos sonidos deben ser introducidos previamente en su memoria. Esta memoria RAM puede ser cargada con bancos de sonidos e instrumentos. Después, mediante MIDI pueden ser leídos esos sonidos y reproducirlos.

Ilustración 10. Sampler modular para rack. Akai profesional.

2.2.6 Controlador MIDI

Tras la creación de los primeros sintetizadores, se pensó como conectarlos entre sí para así reproducir más sonidos y crear instrumentos con sonidos más ricos. Con las posibilidades de la comunicación digital se creó un protocolo de comunicación estandarizado para conectar dispositivos que lo incluían. Así apareció el MIDI (Musical Instruments Digital Interface): Interface digital para instrumentos musicales.

Cualquier dispositivo que incluía la especificación MIDI disponía de la posibilidad de conectarse con otro dispositivo MIDI mediante una serie de conectores y unos cables especiales.

El MIDI provocó que muchos de los dispositivos que antes incluían teclado, generador u osciladores de sonido, efectos, etc. se separaran. Esto permitía mayor modularidad y un decremento en el precio de los dispositivos. Luego ya se unirían mediante los cables formando un sistema.

Los cables tienen un conector de tipo DIN de 5 contactos y los dispositivos pueden disponer de diferentes conectores.

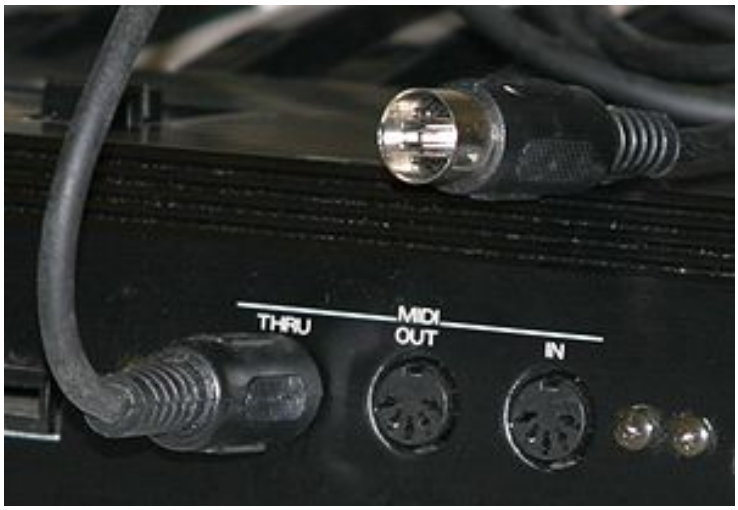


Ilustración 11. Conectores MIDI.

Según la Ilustración 11 podemos ver los diferentes conectores de entrada o salida que puede tener un dispositivo MIDI. MIDI In permite conectar otros dispositivos MIDI de entrada. MIDI Out permite conectar dispositivos de salida a los que van los datos después de ser procesados en el dispositivo. MIDI Thru permite una conexión directa entre el dispositivo que esté conectado en MIDI In a la salida que esté conectada en MIDI Out, es decir, los datos que entren por MIDI In se copian a MIDI Thru[7].

Fueron apareciendo los distintos dispositivos MIDI:

- Teclados MIDI: Dispositivo de datos de entrada. Al carecer, normalmente, de generado y/o procesado de sonido, se llaman también teclados maestros. Solo generan notas y datos MIDI.
- Sintetizador: generador de sonidos por síntesis. Dispositivo de salida.
- Módulos sampler: generador de sonidos por tabla de ondas. Dispositivo de salida.
- MIDI merger
- MIDI patchbay
- Pads de percusión. Dispositivo de datos de entrada.
- Guitarras MIDI. Dispositivo de datos de entrada.
- Violines MIDI. Dispositivo de datos de entrada.
- Instrumentos de viento MIDI. Dispositivo de datos de entrada.
- Secuenciadores. Dispositivo para grabar los datos de entrada MIDI, editarlos y enviarlos hacia otros dispositivos MIDI de salida.
- Controlador MIDI. Dispositivo genérico de datos de entrada.

Ilustración 12. Teclado maestro MIDI Behringer UMX25.

2.2.7 Música por ordenador. Tarjetas de sonido.

Gracias a los conversores analógico-digital y digital-analógico, es posible digitalizar sonido con micrófonos y reproducir sonido con ordenadores mediante altavoces, respectivamente. De esta manera se iniciaron las posibilidades multimedia en informática.

Hoy en día todos los ordenadores incluyen una tarjeta de sonido con múltiples entradas y salidas. Además los sistemas operativos incluyen la gestión de ésta de forma predeterminada.

Las tarjetas de sonido en general disponen de un módulo encargado del audio y otro módulo encargado del MIDI. La parte audio dispone de las entradas y salidas (conversores D/A y A/D), módulos de efectos DSP, etc. La parte MIDI comprende los conectores de entrada y de salida MIDI.

2.2.8 Virtualización de dispositivos audio y MIDI

La evolución de los ordenadores ha permitido que ciertos dispositivos se puedan transformar de electrónicos a virtuales, de hardware a software. Algunos sintetizadores y secuenciadores pueden encontrarse en versión software, así como los controladores MIDI. Todos estos con la versatilidad que proporciona la programación por ordenador. Este paso ha permitido que en un mismo ordenador convivan los mismos tipos de dispositivos que se pueden encontrar en versión hardware pero en versión software.

Dispositivos que podemos encontrar virtualizados:

- Secuenciadores: Como ya he dicho permite la captura, edición y reproducción de datos MIDI pero actualmente además permite la captura de audio, su secuenciación e incluso la posibilidad de aplicar efectos sin salir de él con la utilización de módulos virtuales de efectos. También permite el uso de instrumentos virtuales.

  Ejemplos: Cubase, ProTools, Live, Reaper, etc.

- Cable MIDI virtualizado, permite conectar los distintos dispositivos MIDI virtualizados.

- Instrumentos virtuales: Como su nombre indica, instrumentos, bien generadores por síntesis virtual bien samplers virtuales con memoria para almacenar muestras y que luego reproducen según la ejecución de la nota musical correspondiente. Disponen de conexiones virtuales MIDI.

- Efectos virtuales: como los instrumentos pero aportan modificaciones al sonido que les entra. Reverberación, delays, filtros, ecualizadores, compresores, limitadores, etc. Cuyos parámetros suelen poder ser controlados via MIDI.

  Tanto los instrumentos como los efectos suelen considerarse como "plugins" para un secuenciador u otros "hosts" de este tipo.

Algunas de las tecnologías utilizadas para diseñar instrumentos y efectos virtuales son:

- VST: Introducida por Steinberg en su producto secuenciador Cubase v3 y considerada un estándar de la industria de este tipo.

- AU: Parte de Core Audio propiedad de Apple y también soportada por otros productos como Ableton Live, Steinberg Cubase, Cakewalk Sonar, etc.

- RTAS: soportado solo por ProTools de Digidesign.

- TDM: versión de ProTools soportado por equipos de alta calidad en estudios.

Estos dispositivos virtuales son programas y su proceso siempre ofrece una cierta latencia tanto si se utilizan grandes cantidades de datos de entrada como si el proceso que genera la salida es complejo.

Los controladores MIDI nos van a ayudar a administrar los parámetros del secuenciador, de los instrumentos y de los plugins de efectos conectados a él.

A continuación se presenta un esquema de las posibles conexiones que pueden definirse y sus dispositivos:

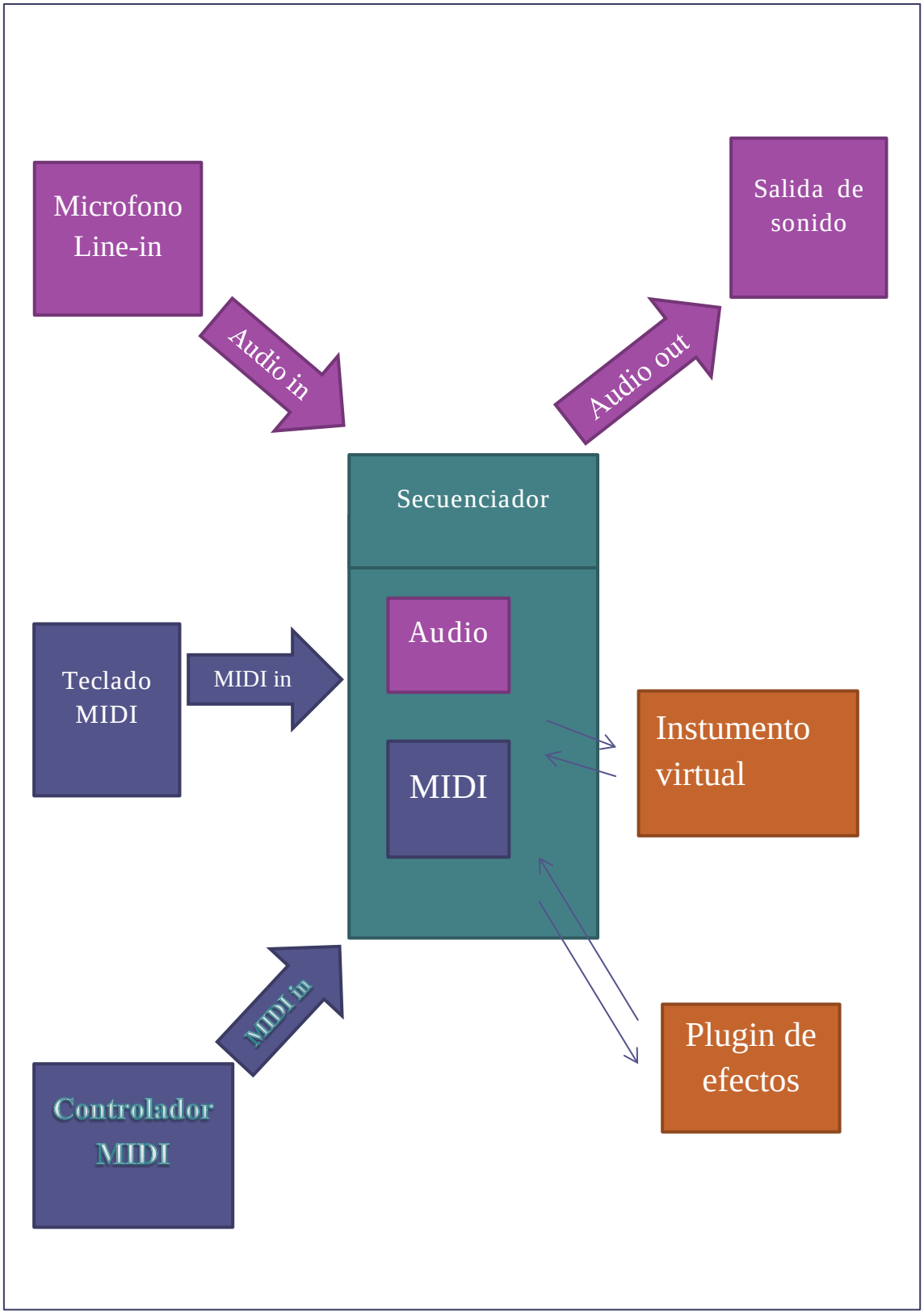


Ilustración 13. Sistema audio/midi virtual.

3 Especificación de requisitos

3.1 Requisitos

He pensado en lo que hace falta para hacer funcionar el proyecto con los programas de terceros necesarios y estas son las conclusiones:

- Necesitaré un software que proporciona cable virtual entre los dispositivos reales para recoger tanto la salida del secuenciador como la salida del controlador. El secuenciador puede reconocer los dispositivos MIDI y también puede enviar y recoger datos entre estos y las pistas de un proyecto y otros dispositivos. Existen muchos, son gratuitos y se conocen como Puertos MIDI virtuales. MIDIYoke, LoopBe1, LoopMIDI. Yo utilizaré **LoopMIDI**[8], es el más moderno, y es gratuito aunque cualquiera de los otros serviría.

- Los drivers del controlador MIDI hardware. Permitirán hacer la entrada al sistema informático y en consecuencia al secuenciador. Son propietarios por lo tanto dependen del hardware. En este caso voy a utilizar la primera versión de **nanoKontrol**, controlador MIDI de la marca KORG.

- Un software secuenciador. Es el anfitrión que encadenará todo, software, hardware mediante el protocolo MIDI. Como ya hemos visto existen varios secuenciadores: Cubase, Ableton Live, Protools, etc. Yo voy a utilizar **Reaper**[9] pues es suficiente para el buen funcionamiento del proyecto, y tiene plena funcionalidad a pesar de ser un programa de prueba si lo descargas y no compras una licencia.

- Y por supuesto el programa que se diseña en este proyecto, **MIDIm**. MIDIm utiliza el protocolo MIDI para recoger valores de un determinado puerto MIDI para luego enviarlos por otro puerto una vez procesados. En la creación del programa se utilizarán una serie de tecnologías de programación que permiten el acceso a dispositivos MIDI. En los siguientes puntos se va a desarrollar las características del protocolo MIDI, es decir, aspectos teóricos, y como programarlo mediante las librerías adecuadas.

3.2 El Protocolo MIDI

MIDI (abreviación para Musical Instrument Digital Interface) es un estándar tecnológico que describe un protocolo, una interface digital y conectores que permiten que varios instrumentos musicales electrónicos, computadoras y otros dispositivos relacionados se conecten y comuniquen entre sí[10].

El protocolo MIDI es un protocolo de comunicación entre dispositivos. Los dispositivos MIDI entienden este lenguaje y se comunican mediante el envío de mensajes entre ellos.

Por ejemplo, imaginemos que disponemos de dos dispositivos MIDI, un teclado controlador MIDI y un sintetizador. Conectamos los dos dispositivos con el cable tal y como se muestra en la Ilustración 14. El teclado dispone de un conector MIDI out y el sintetizador un conector MIDI in. Cuando pulsamos una tecla en el teclado se envía un mensaje por el cable que es interpretado por el sintetizador y este reproduce un sonido.

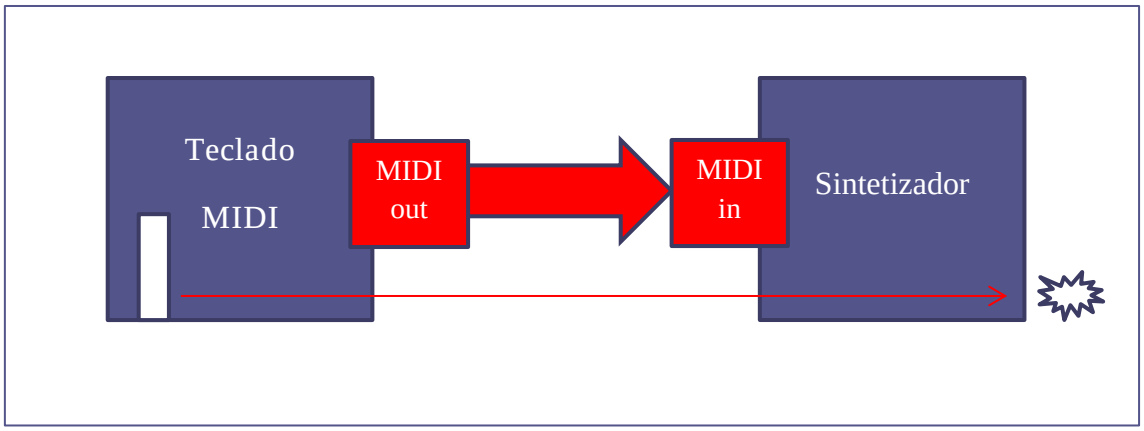


Ilustración 14. Conexión MIDI entre teclado y sintetizador.

El protocolo MIDI es un protocolo serie, entre dos puntos. Pues bien, mediante el conector MIDI thru es posible conectar más dispositivos como destino de los mensajes. Como puede comprobarse en la Ilustración 15, el mensaje será recibido por el sintetizador y por el sampler. Los dos podrán ejecutar sonido como respuesta al evento de llegada del mensaje enviado por el teclado.

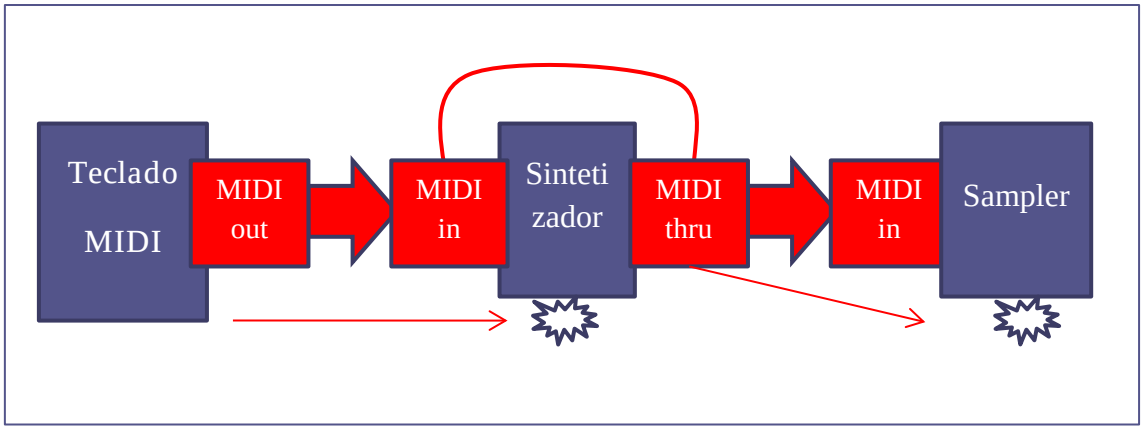


Ilustración 15. MIDI thru

Como vemos, la forma de controlar los dispositivos MIDI es realizada mediante el envío de mensajes. Pero ¿Cómo se controla más de un dispositivo MIDI?, ¿Cómo son los mensajes?, ¿De qué tipo son?. Vamos a responder a todas estas preguntas.

3.2.1 Los mensajes MIDI

Los mensajes MIDI se clasifican en mensajes de canal y mensajes de sistema.

Los mensajes de canal se aplican al canal especificado en el mensaje. Cada dispositivo MIDI puede controlar hasta 16 canales. Los mensajes de canal están clasificados en Note Off, Note On, Poly. Aftertouch, Control Change, Chan. Aftertouch, Pitch Bend y Program Change. Más tarde los analizaremos, sobre todo los de control.

En este tipo de mensajes es necesario seleccionar el canal que va a utilizar el dispositivo y será por el que enviará o en el que recibirá mensajes de canal.

Por ejemplo, en un teclado MIDI se puede seleccionar por qué canal va a enviar las notas. Si elegimos el canal 1 en el teclado los otros dispositivos que estén esperando mensajes de canal 1 interpretarán los mensajes. El resto de dispositivos escucharán también pero no harán caso de estos mensajes dirigidos al canal 1 aunque estén conectados fisicamente. Si en el teclado cambiamos al canal 2, los subsiguientes mensajes irán destinados a los dispositivos MIDI que estén esperando mensajes en este canal.

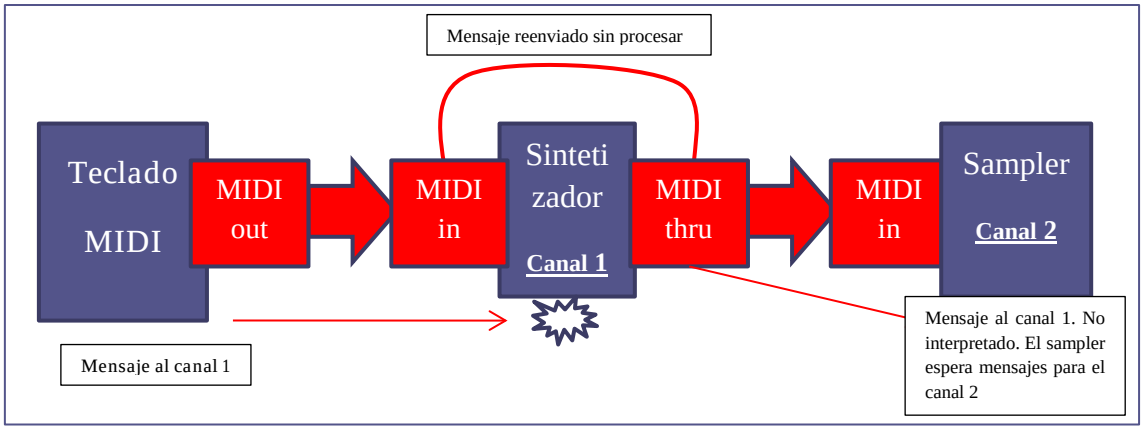


Ilustración 16. Canales MIDI.

Cuando utilizamos un secuenciador, sobre todo si es por software, cada pista que va a almacenar/reproducir datos MIDI tiene seleccionado un canal MIDI determinado. El teclado suele estar configurado por defecto para enviar mensajes MIDI en el canal 1.

Ilustración 17. Pistas MIDI

Entonces podemos hacer que el secuenciador cambie el canal en el mensaje cuando seleccionamos que canal va a escuchar la entrada MIDI . En la Ilustración 17 podemos ver que los mensajes del teclado son del canal 1 pero el secuenciador los traducirá al canal 3 puesto que la escucha está seleccionada en la pista que está configurada con este canal. Como podemos observar los secuenciadores proporcionan una cierta flexibilidad tanto para la entrada de datos MIDI como para el envío de estos.

Los mensajes de sistema no tienen asociado un canal y son de tres tipos: mensajes comunes del sistema como MIDI Time Code (MTC), mensajes de sistema de tiempo real que se utilizan para sincronizar dispositivos y mensajes de sistema exclusivo que permite a los fabricantes de dispositivos agrandar sus características de forma no estándar.

Vamos a centrarnos en los mensajes de canal y su estructura interna.

En general los mensajes MIDI tienen un tamaño mínimo de dos bytes y un tamaño máximo de tres bytes. El primero se llama de estado (STATUS BYTE) y los dos siguientes de datos (DATA 1 y DATA 2). Ver Ilustración 18. El tercer byte de datos es opcional, es decir, que hay mensajes en los que tiene sentido y se utiliza, y otros en los que no se utiliza.

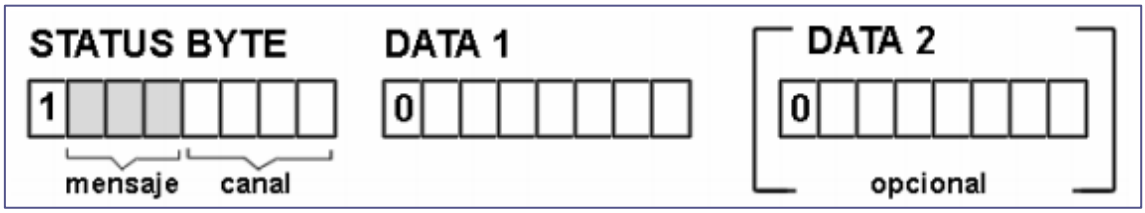


Ilustración 18. Mensaje MIDI.

El byte de estado siempre empieza con un bit a 1. Los siguientes 3 bits representan el tipo de mensaje y los cuatro siguientes al número de canal. De este byte se puede deducir que el máximo de canales son 16 ya que 2 valores posibles en cada una de las 4 posiciones: $2^4 = 16$. Del 0000 al 1111 que representan al canal 1 y 16 respectivamente. Los de datos siempre empiezan con el primer bit a 0 por lo tanto solo se permiten 128 valores distintos (de 0 al 127) en cada byte de datos. Es decir, del 00000000 al 01111111. De aquí se deduce que ciertos datos y controles tengan un valor que siempre está dentro de ese intervalo de valores.

Byte de estado		Datos 1	Datos 2	Explicación
Tipo de mensaje	Denominación			
000	Note Off	Tono	Volumen	Dejar de reproducir nota
001	Note On	Tono	Volumen	Reproducir nota con tono y volumen
010	Poly Aftertouch	Altura	Presión	Presión polifónica
011	Control Change	Control	Valor	Cambio de control
100	Channel Aftertouch	Presión		Presión pero por canal
101	Pitch Bend	MSB	LSB	Microcambio tonal
110	Program Change	Programa		Tipo de sonido

Ilustración 19. Descripción de los mensajes MIDI.

En cambio, en el tipo de mensaje Pitch Bend se combinan los bytes de datos para formar un único dato con su byte menos significativo (LSB) y su byte más significativo (MSB) de forma que se obtienen valores representados por 14 bits, $2^{14} = 16384$ (del -8192 al 8191)[11].

En la Ilustración 19 podemos ver los tipos de mensajes de canal disponibles. Los de tipo “Note On” y “Note Off” se utilizan para hacer sonar notas y pararlas en el tiempo con una cierta tonalidad y un cierto volumen. Los de tipo “Aftertouch” son interpretados por el dispositivo para que cambie el tipo de presión que se haría sobre el instrumento al tocarlo (presión en la tecla o cuerda, etc.). Hay mensajes para seleccionar el tipo de sonido (el instrumento seleccionado).

Con “Program Change” podemos seleccionar uno de los instrumentos que el dispositivo puede ejecutar. Como se puede deducir fácilmente (solo se usa un byte de datos) cada dispositivo puede tener como máximo 128 instrumentos.

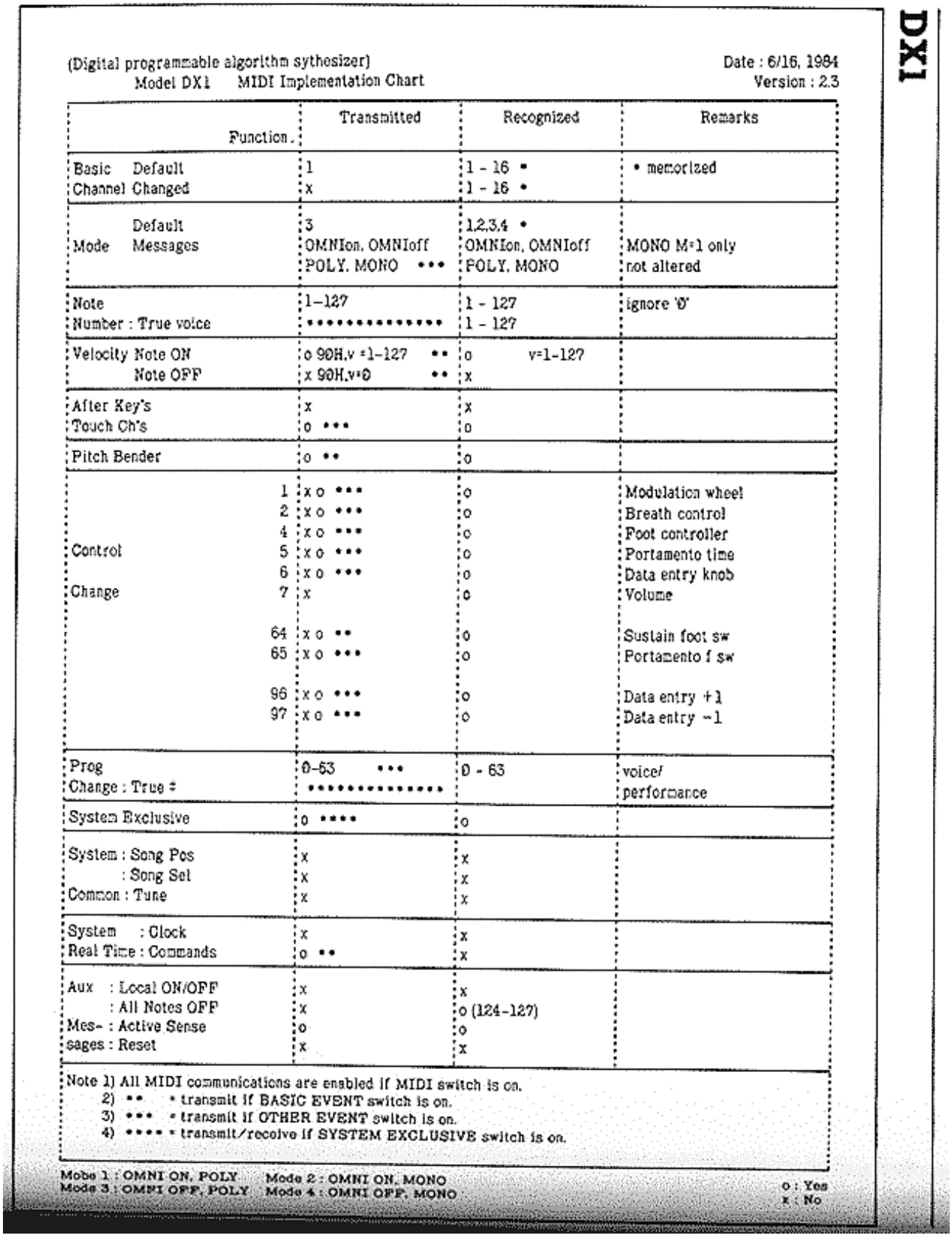
Y por último, “Control Change”, es un tipo de mensaje que permite cambiar alguna de las características en la ejecución del instrumento. El primer byte define uno de los 128 tipos de controles. El segundo byte, define el valor para ese control especificado. Por ejemplo, el control 7 suele corresponder al volumen del canal, el control 1 cambia la modulación, los controles 91 y 93 suelen corresponder con la aplicación del efecto de reverberación y coro respectivamente, etc.

Es importante el hecho que no es obligado que el dispositivo tenga esos controles ubicados con los mismos códigos, es decir, el control 7 puede corresponder a otra característica distinta en otro dispositivo. Lo que ocurre es que se considera un estándar el hecho de utilizar siempre los mismos números. De todas formas cualquier dispositivo siempre debe ir acompañado de su "hoja de especificación MIDI" en la que aparezcan los controles disponibles.

En la Ilustración 20 se puede ver lo que sería la hoja de especificaciones o implementación MIDI[12] del sintetizador Yamaha DX1. Como es un dispositivo de salida, este reconoce ciertos controles (ver columna "recognized" en la Ilustración 20) muchos de ellos estándar. Lo mismo ocurre con "Program Change", hay un estándar General MIDI (GM) que especifica el uso de cada uno de los códigos para el byte de datos 1. En la Ilustración 21 se puede ver como este estándar especifica una lista de instrumentos a los que corresponden los códigos de programa.

Si un dispositivo, como un módulo de sonido, cumple el estándar GM tendrá todos los instrumentos mencionados en la Ilustración 21 y podrá ser seleccionado con el código asociado en un mensaje de "Program Change" (en binario: 1110 0000, 0111 1111, es decir, "Program Change" del canal 1 seleccionando el programa 127 gun shot). Ojo, de 1 a 128 en la lista corresponde al dato de 0 a 127.

El proyecto que nos ocupa se encargará de escuchar los controles MIDI que vienen del dispositivo de entrada (controlador MIDI) y que vienen del secuenciador para memorizarlos, modificarlos según sea el caso y reenviarlos de nuevo al secuenciador. Ver de nuevo la Ilustración 3.



DX1

(Digital programmable algorithm sythesizer)
Model DX1 MIDI Implementation Chart

Date : 6/16, 1984
Version : 2.3

Function	Transmitted	Recognized	Remarks
Basic Default	1	1 - 16 *	* memorized
Channel Changed	x	1 - 16 *	
Mode Default	3	1,2,3,4 *	
Mode Messages	OMNIon, OMNIoff	OMNIon, OMNIoff	MONO M=1 only
	POLY, MONO ***	POLY, MONO	not altered
Note Number : True voice	1-127 **********	1 - 127 1 - 127	ignore '0'
Velocity Note ON	o 90H,v =1-127 **	o v=1-127	
Note OFF	x 90H,v=0 **	x	
After Key's	x	x	
Touch Ch's	o ***	o	
Pitch Bender	o **	o	
Control Change 1	x o ***	o	Modulation wheel
2	x o ***	o	Breath control
4	x o ***	o	Foot controller
5	x o ***	o	Portamento time
6	x o ***	o	Data entry knob
7	x	o	Volume
64	x o **	o	Sustain foot sw
65	x o ***	o	Portamento f sw
96	x o ***	o	Data entry +1
97	x o ***	o	Data entry -1
Prog Change : True #	0-63 *** **********	0 - 63	voice/ performance
System Exclusive	o ****	o	
System : Song Pos	x	x	
: Song Sel	x	x	
Common : Tune	x	x	
System : Clock	x	x	
Real Time : Commands	o **	x	
Aux : Local ON/OFF	x	x	
: All Notes OFF	x	o (124-127)	
Mes- : Active Sense	o	o	
sages : Reset	x	x	

Note 1) All MIDI communications are enabled if MIDI switch is on.
2) ** * transmit if BASIC EVENT switch is on.
3) *** * transmit if OTHER EVENT switch is on.
4) **** * transmit/receive if SYSTEM EXCLUSIVE switch is on.

Mode 1 : OMNI ON, POLY Mode 2 : OMNI ON, MONO
Mode 3 : OMNI OFF, POLY Mode 4 : OMNI OFF, MONO

o : Yes
x : No

Ilustración 20. Especificaciones MIDI del sintetizador Yamaha DX1.

GENERAL MIDI PATCH MAP

1. Acoustic grand piano
2. Bright acoustic piano
3. Electric grand piano
4. Honky-tonk piano
5. Rhodes piano
6. Chorused piano
7. Harpsichord
8. Clavinet
9. Celesta
10. Glockenspiel
11. Music box
12. Vibraphone
13. Marimba
14. Xylophone
15. Tubular bells
16. Dulcimer
17. Hammond organ
18. Percussive organ
19. Rock organ
20. Church organ
21. Reed organ
22. Accordion
23. Harmonica
24. Tango accordion
25. Acoustic nylon guitar
26. Acoustic steel guitar
27. Electric jazz guitar
28. Electric clean guitar
29. Electric muted guitar
30. Overdriven guitar
31. Distortion guitar
32. Guitar harmonics
33. Acoustic bass
34. Electric bass fingered
35. Electric bass picked
36. Fretless bass
37. Slap bass 1
38. Slap bass 2
39. Synth bass 1
40. Synth bass 2
41. Violin
42. Viola
43. Cello
44. Contrabass
45. Tremolo strings
46. Pizzicato strings
47. Orchestral harp
48. Timpani
49. String ensemble 1
50. String ensemble 2
51. Synth strings 1
52. Synth strings 2
53. Choir aahs
54. Voice oohs
55. Synth voice
56. Orchestra hit
57. Trumpet
58. Trombone
59. Tuba
60. Muted trumpet
61. French horn
62. Brass section
63. Synth brass 1
64. Synth brass 2
65. Soprano sax
66. Alto sax
67. Tenor sax
68. Baritone sax
69. Oboe
70. English horn
71. Bassoon
72. Clarinet
73. Piccolo
74. Flute
75. Recorder
76. Pan flute
77. Bottle blow
78. Shakuhachi
79. Whistle
80. Ocarina
81. Square wave lead
82. Sawtooth wave lead
83. Caliope lead
84. Chiff lead
85. Charang
86. Solo synth voice
87. Bright saw wave lead
88. Brass and lead
89. Fantasia pad
90. Warm pad
91. Poly synth pad
92. Space voices pad
93. Bowed glass pad
94. Metal pad
95. Halo pad
96. Sweep pad
97. Ice rain
98. Soundtrack
99. Crystal
100. Atmosphere
101. Brightness
102. Goblin
103. Echo drops
104. Star theme
105. Sitar
106. Banjo
107. Shamisen
108. Koto
109. Kalimba
110. Bag pipe
111. Fiddle
112. Shanai
113. Tinkle bell
114. Agogo
115. Steel drums
116. Woodblock
117. Taiko drum
118. Melodic tom
119. Synth drum
120. Reverse cymbal
121. Guitar fret noise
122. Breath noise
123. Sea shore
124. Bird tweet
125. Telephone ring
126. Helicopter
127. Applause
128. Gun shot

Ilustración 21. General MIDI. Program change.

3.3 El soporte multimedia en Windows

Windows nos proporciona dos métodos para la programación multimedia: Uno es el acceso mediante comandos MCI y el otro mediante las librerías multimedia que forman parte del API de Windows. Toda la documentación sobre el API de Windows se puede encontrar en el MSDN de Microsoft[13].

Los comandos MCI (Media Control Interface) permiten la gestión de los distintos dispositivos multimedia que se pueden controlar en el sistema. Los comandos son los mismos para la todos los dispositivos lo cual facilita la programación de cada uno de los distintos dispositivos pero por otro lado, esta generalización, no permite un control personalizado de las características propias del dispositivo. Cuando decimos dispositivos multimedia decimos, por

ejemplo, CDaudio, Video digital, Captura de video, Escáner de imágenes, Secuenciador MIDI, Archivos wav, etc.

Todos se pueden gestionar con estos comandos identificando el tipo de dispositivo y el tipo de acción a realizar mediante cadenas de texto. Existen comandos para todos los dispositivos que permiten abrir, cerrar y empezar la reproducción con las cadenas “open”, “close”, y “play”. Los dispositivos también se representan con cadenas de texto, por ejemplo, “cdaudio” para reproducir un compact disk de audio, “sequencer” para reproducir MIDI o “waveaudio” para reproducir un fichero wav, etc.

Mediante un “MCI command string” nos comunicamos con el dispositivos diciéndole lo que queremos que haga. Por ejemplo, enviándole “play cdaudio” se pondría a reproducir el cd de audio, o “pause cdaudio” se pararía la reproducción.

En nuestro caso, el método mediante comandos MCI no soluciona nuestro problema de recoger los datos de entrada ni de la salida de los dispositivos MIDI. Es decir, de manejar mensajes MIDI y extraer los datos internos. Para ello utilizaremos las funciones del API multimedia de Windows.

El segundo método de que dispone Windows para programar con dispositivos multimedia, más concretamente MIDI, es mediante su API. Windows proporciona una librería DLL, winmm.dll, con funciones que permiten programar tanto con comandos MCI como el acceso a bajo nivel de los dispositivos multimedia.

En el caso de MIDI nos proporciona las distintas funciones enumeradas a continuación:

- **midiConnect**
- **midiDisconnect**
- **midiInAddBuffer**
- **midiInClose**
- **midiInGetDevCaps**
- **midiInGetErrorText**
- **midiInGetID**
- **midiInGetNumDevs**
- **midiInMessage**
- **midiInOpen**
- **midiInPrepareHeader**
- **MidiInProc**
- **midiInReset**
- **midiInStart**
- **midiInStop**
- **midiInUnprepareHeader**
- **midiOutCacheDrumPatches**
- **midiOutCachePatches**
- **midiOutClose**
- **midiOutGetDevCaps**
- **midiOutGetErrorText**
- **midiOutGetID**
- **midiOutGetNumDevs**
- **midiOutGetVolume**
- **midiOutLongMsg**
- **midiOutMessage**
- **midiOutOpen**
- **midiOutPrepareHeader**
- **MidiOutProc**
- **midiOutReset**
- **midiOutSetVolume**
- **midiOutShortMsg**
- **midiOutUnprepareHeader**
- **midiStreamClose**

- midiStreamOpen
- midiStreamOut
- midiStreamPause
- midiStreamPosition
- midiStreamProperty
- midiStreamRestart
- midiStreamStop

Y las estructuras de datos siguientes:

- MIDIEVENT
- MIDIHDR
- MIDIINCAPS
- MIDIOUTCAPS
- MIDIPROPTEMPO
- MIDIPROPTIMEDIV
- MIDISTRMBUFFVER

De todas las funciones anteriores vamos a descubrir el significado de las que se van a utilizar en el proyecto así como el formato de los datos, las estructuras de datos y el método de programación. Para más información no contenida en este punto es necesario acudir a MSDN de Microsoft o libros sobre el API de multimedia de Windows. En esta memoria solo se comentará la información más relevante para la comprensión del proyecto.

Para saber qué número de dispositivos MIDI hay conectados en el sistema se utilizan las funciones *midiInGetNumDevs* y *midiOutGetNumDevs* los cuales devuelven el número de dispositivos MIDI de entrada y de salida, respectivamente. Estas funciones devuelven un valor entero de 16 bits sin signo que puede tener un valor mayor o igual a 0 que significa el número de dispositivos en cada caso.

```
MMRESULT midiInGetDevCaps(
   UINT_PTR      uDeviceID,
   LPMIDIINCAPS lpMidiInCaps,
   UINT          cbMidiInCaps
);
```

```
MMRESULT midiOutGetDevCaps(
   UINT_PTR      uDeviceID,
   LPMIDIOUTCAPS lpMidiOutCaps,
   UINT          cbMidiOutCaps
);
```

Ilustración 22. Obtener capacidades de los dispositivos MIDI.

Una vez averiguado el número de dispositivos es posible saber algunas características del dispositivo como el número de índice en el sistema o el nombre que se le ha atribuido, etc. Para ello se utiliza sobre los dispositivos de entrada y los de salida las funciones *midiInGetDevCaps* y *midiOutGetDevCaps*, respectivamente. Ver Ilustración 22. Estas funciones requieren de 3 parámetros obligatorios. El primero de ellos el índice en el sistema (puede ser de 0 a el número de dispositivos obtenidos con las funciones *midiInGetNumDevs* o *midiOutGetNumDevs* menos 1). El segundo parámetro es un puntero a una variable con la estructura *MIDIINCAPS* o *MIDIOUTCAPS* en su caso.

```
typedef struct {
  WORD      wMid;
  WORD      wPid;
  MMVERSION vDriverVersion;
  TCHAR     szPname[MAXPNAMELEN];
  DWORD     dwSupport;
} MIDIINCAPS;

typedef struct {
  WORD      wMid;
  WORD      wPid;
  MMVERSION vDriverVersion;
  TCHAR     szPname[MAXPNAMELEN];
  WORD      wTechnology;
  WORD      wVoices;
  WORD      wNotes;
  WORD      wChannelMask;
  DWORD     dwSupport;
} MIDIOUTCAPS;
```

Ilustración 23. Capacidad de los dispositivos MIDI.

En la estructura *MIDIINCAPS*, el miembro más importante es la cadena de texto *szPname* que almacena el nombre del dispositivo de entrada. Con respecto a la estructura *MIDIOUTCAPS*, también se puede obtener el nombre del dispositivo además de ciertos datos como el número de voces que soporta el dispositivo, el tipo de sintetizador (si es normal, FM, de tabla de ondas o por software, etc.) y otras características menos relevantes tal y como puede verse en la Ilustración 23.

Por último, el tercer parámetro corresponde al tamaño de la variable que contendrá los datos estructurados.

```
MMRESULT midiInOpen(
   LPHMIDIIN lphMidiIn,
   UINT      uDeviceID,
   DWORD_PTR dwCallback,
   DWORD_PTR dwCallbackInstance,
   DWORD     dwFlags
);
```

Ilustración 24. Abrir dispositivos MIDI

En todo caso, es necesario abrir el dispositivo para poder gestionarlo y después cerrarlo como si de un fichero se tratara. Esto mismo se realiza con las

funciones *midiInOpen*, *midiOutOpen*, *midiInClose* y *midiOutClose*. Ver Ilustración 24.

```
void CALLBACK MidiInProc(
   HMIDIIN   hMidiIn,
   UINT      wMsg,
   DWORD_PTR dwInstance,
   DWORD_PTR dwParam1,
   DWORD_PTR dwParam2
);
```

Ilustración 25. Función CallBack.

Estas funciones requieren de algunos parámetros comunes como el identificativo del dispositivo a ser abierto o cerrado en su caso. Más concretamente en el caso de la apertura de un dispositivo para la entrada de datos MIDI (*midiInOpen*) es necesario especificar un puntero a una función que se encargará de recoger los datos proporcionados por el sistema desde el dispositivo. Este tipo de funciones son del tipo CALLBACK_FUNCTION (funciones de retrollamada) que es otro de los parámetros que hay que especificar en la apertura del dispositivo de entrada. Esta función la define el usuario según el modelo de la Ilustración 25 y será la que procesa los datos que recibe del dispositivo de entrada cada vez que este los envíe al sistema. Más adelante veremos cómo se han implementado dichas funciones "callback" en la solución que se propone en el proyecto mediante el programa MIDIm. También como se manejan los datos MIDI de los dispositivos de entrada y como se preparan para la salida hacia los dispositivo de salida.

Una vez abierto el dispositivo ya estamos en disposición de recibir datos a través de la función de "callback" definida en el uso de la función *midiInOpen* y de enviar datos con la función *midiOutShortMsg* tal y como se define en la Ilustración 26.

```
MMRESULT midiOutShortMsg(
   HMIDIOUT hmo,
   DWORD    dwMsg
);
```

Ilustración 26. Enviar datos MIDI.

El primer parámetro *hmo* es el manejador que identifica al dispositivo al que se le va a enviar el parámetro *dwMsg*. Este último parámetro contiene los bytes del mensaje de acuerdo al formato de mensaje MIDI tal y como se muestra en la Ilustración 18. El parámetro *dwMsg* es valor de 4 bytes. Los 3 primeros bytes empezando por la parte baja corresponden al byte de estado, el byte de

primer dato MIDI y el byte del segundo dato MIDI en el caso de que se usen. En el caso del programa se usarán los tres pues estamos esperando valores de controles MIDI.

En la parte de codificación del programa se verá cómo definir las funciones y como rescatar los datos del mensaje MIDI así como la elaboración de este para el lenguaje C#.

Existen en el API otras funciones de tipo “stream” que no han sido utilizadas en el proyecto por lo que se remite al lector a que las descubra en la documentación de Microsoft antes mencionada.

4 Diseño e implementación

En este punto nos vamos a encargar de revisar el diseño e implementación del programa MIDIm expuesto en este proyecto.

Se ha elegido el sistema operativo Windows como sistema final para el desarrollo y la ejecución del programa. Hay varias razones del porque se ha elegido este sistema pero la más importante es que tiene un buen soporte para multimedia en general y MIDI en particular. Como hemos visto en el punto anterior, el sistema nos provee de librerías que permiten controlar los dispositivos MIDI y los mensajes entre ellos. Por otro lado Windows dispone de un gran soporte de “drivers” que ofrecen los fabricantes de dispositivos, sobre todo en multimedia. En otros sistemas podríamos haber tenido más problemas de reconocimiento o de gestión de los dispositivos hardware por parte del sistema. Otra de las razones del sistema operativo Windows es su sencillez de manejo y universalidad.

Es posible programar directamente con las librerías que proporciona de forma nativa Windows mediante el lenguaje de programación C++ dando lugar a un código pequeño y rápido aunque nativo, es decir, no portable a otros sistemas.

Existe también la posibilidad de programar en java. Java proporciona también soporte multimedia y MIDI, y sobre todo es portable a otros sistemas. En este caso no se ha elegido esta tecnología porque lo que se pretendía era utilizar las librerías más rápidas en ejecución de la gestión MIDI disponibles para Windows. En java la ejecución es interpretada mediante una máquina virtual intermedia mientras el acceso que se hace a las librerías del sistema es código nativo ejecutado por el sistema directamente.

Sin embargo, de todas las posibilidades que ofrece o permite Windows para el desarrollo de aplicaciones, se ha elegido .net framework. Este provee, de forma sencilla, de un marco en la programación para Windows y de una programación orientada al objeto. .Net nos provee de una gran cantidad de librerías de clases ya definidas. En este caso se ha codificado en C# y .net framework que es suficiente para nuestro propósito de gestionar una ventana y algunos controles típicos de Windows. En cuanto al acceso a las librerías nativas de multimedia, .net permite el acceso nativo a ella como se ha visto en el punto anterior.

También se podía haber programado en otro lenguaje soportado por .net framework (como visual basic, etc.) aunque realmente da lo mismo pues el código generado o código administrado (como se le denomina) para la ejecución final sería el mismo.

El IDE (entorno de programación) utilizado podría haber sido Visual Studio de Microsoft aunque se ha elegido Sharpdevelop[14] el cual es gratuito. Con Sharpdevelop se pueden hacer gran parte de las acciones básicas que son posibles en Visual Studio. La apariencia es similar pero dispone de menos posibilidades. En la fase de desarrollo del programa es posible realizar (con Sharpdevelop) programación en tiempo de diseño de la ventana, controles, colores e incluso ayuda en el control de los eventos. También dispone de funciones de completar código mostrando métodos y propiedades de las clases utilizadas a medida que se va escribiendo código.

4.1 Abstración

El programa MIDIm consiste en recoger datos MIDI (datos de control más concretamente) de los dos dispositivos implicados memorizándolos en una matriz.

Cuando el dato que recoge MIDIm es del controlador MIDI hardware este es enviado con o sin modificación (según el modo seleccionado) al secuenciador. En el caso de que se modifique el dato, el dato es calculado teniendo en cuenta los valores leídos con los anteriores memorizados. Cuando el dato proviene del secuenciador este solo se memoriza. Ver Ilustración 1, Ilustración 2 e Ilustración 3.

En cuanto a la abstracción en pseudocódigo el programa consiste como en tres partes que son gestionados completamente por el sistema. La primera o principal se encarga de abrir y cerrar dispositivos, representar en pantalla y atender a los eventos de la ventana y controles Windows y definir las funciones “callback” de entrada de datos MIDI. A continuación se presenta la primera parte:

1. *Leer configuración*
2. *En el caso de modo_seleccionado*
    a. *Caso normal: establecer callback_normal*
    b. *Caso buscar: establecer callback_buscar*
    c. *Caso proporcional: establecer callback_proporcional*
3. *Abrir dispositivo_entrada_secuenciador(callback_secuenciador)*
4. *Abrir dispositivo_entrada_controlador(modo_seleccionado)*
5. *Abrir dispositivo_salida_secuenciador*
6. *Esperar cierre_aplicacion*
    a. *Grabar configuración*
    b. *Cerrar dispositivo_**
    c. *Terminar_programa*

La siguiente parte procesa los datos MIDI que recibe del controlador. Y los envía procesados al secuenciador. Corresponde a la función de “callback” establecida en modo_seleccionado:

1. *Recibe dato*
2. *Si no es mensaje_ de_canal entonces salir*
3. *Establecer control_numero=byte correspondiente de dato*
4. *Establecer control_valor=byte correspondiente de dato*
5. *Si no es mensaje_de_control entonces salir*
6. *Leer anterior=array(control_numero)*
7. *Establecer control_valor=procesar ([normal, buscar o proporcional] , anterior)*
8. *Establecer array(control_numero)=control_valor*
9. *Enviar control_valor*
10. *Salir*

La última parte procesa los datos MIDI pero esta vez del secuenciador. Corresponde a la función callback_secuenciador:

1. *Recibe dato*
2. *Si no es mensaje_ de_canal entonces salir*
3. *Establecer control_numero=byte correspondiente de dato*
4. *Establecer control_valor=byte correspondiente de dato*
5. *Si no es mensaje_de_control entonces salir*
6. *Establecer array(control_numero)=control_valor*

En la segunda parte hay una función llamada *procesar* que representa a los tres tipos de cálculos del valor a enviar de nuevo al secuenciador:

- Normal: aquí no realiza ningún cálculo y envía el dato sin tener en cuenta el dato del secuenciador actual.

- Buscar: En este caso se realiza el cálculo que consiste en no enviar nada hasta que el valor de control leído del controlador MIDI coincide (en un intervalo) con el valor del secuenciador

- Proporcional: Por último, se calcula el valor a enviar teniendo en cuenta cuanto queda para llegar a los bordes, es decir, al 0 o al 127 en el control determinado.

4.2 Codificación

En la fase de codificación se ha utilizado C Sharp (C#) con las librerías .net Framework y con acceso a las librerías multimedia MIDI proporcionadas por el API de Windows de la librería DLL winmm.dll.

El programa consta de 4 clases principales.

- La primera, Programa.cs, es la que contiene el método main principal y ha sido proporcionada completamente al crear el proyecto en el entorno de programación SharpDevelop. No tiene más relevancia que la cargar la clase MainForm para que se ejecute la ventana del programa.

- La segunda de ellas, MainForm.cs, corresponde a la Ventana Windows y sus componentes. Ha sido proporcionada por el entorno de programación en la fase de diseño de la Ventana y controles. Esta clase se encarga de instanciar objetos del resto de clases, controlar los eventos de la ventana, controles, etc.

- La siguiente, win32.cs, es una clase que podríamos considerar estática o por lo menos es una clase que no se instancia sino que dispone de funciones estáticas para el uso inmediato. Contiene algunas de las funciones de acceso desde C# al API de la librería winmm.dll para el caso de las funciones y estructuras MIDI las cuales son utilizadas en el programa. En el siguiente punto analizaremos más detenidamente esta clase.

- La última, Midiio.cs, inicializa, gestiona, todo el potencial MIDI del programa. Más adelante nos encargaremos de ella.

4.2.1 La clase win32

La clase win32 es una clase que no hay que instanciar. Esta clase provee del acceso, con C# .net framework, a las funciones MIDI del API de Windows. Más concretamente a las funciones definidas en la librería dinámica de Windows winmm.dll.

En la programación nativa de Windows se utiliza C/C++ como lenguaje recomendado. El SDK de Windows nos proporciona la definición de constantes, estructuras de datos, las cabeceras de las funciones y las librerías necesarias para la programación. En C/C++ las tres primeras son definidas en los ficheros con extensión “.h”, las librerías con las funciones definidas en los ficheros con extensión “.lib” y el código fuente del usuario en ficheros “.c” y “.cpp”, para programar en C y C++, respectivamente. Por ejemplo, si se programa en C/C++ y se quieren utilizar las funciones MIDI antes descritas es necesario decírselo al compilador. Es necesario incluir la cabecera a usar en forma de texto “#include <mmsystem.h>” al principio del fichero .c o .cpp. De esa forma el compilador ya sabe la definición de constantes, estructuras de datos y funciones. Al compilar y enlazar habrá que decir en que librería buscar el código objeto de estas funciones: la librería “Winmm.lib”. Más tarde en la fase de ejecución se

hará uso de estas funciones cargando la librería dinámica “Winmm.dll” en memoria.

```
using System;
using System.Runtime.InteropServices;

namespace MIDIm
{
    class win32
    {
        public const int CALLBACK_FUNCTION = 0x30000;
        public const int MM_MIM_DATA = 0x3C3;
        public const int MIM_DATA = MM_MIM_DATA;
        public const int MAXPNAMELEN = 32;

        public delegate void MidiInProc(
            int hMidiIn,
            int wMsg,
            int dwInstance,
            int dwParam1,
            int dwParam2
        );

        [StructLayout(LayoutKind.Sequential)]
        public struct MIDIHDR {
            internal Byte[] lpData;
            internal int dwBufferLength;
            internal int dwBytesRecorded;
            internal int dwUser;
            internal int dwFlags;
            internal int lpNext;
            internal int Reserved;
        }

        [StructLayout(LayoutKind.Sequential)]
        public struct MIDIOUTCAPS
        {
            internal Int16 wMid;
            internal Int16 wPid;
            internal int vDriverVersion;
            [MarshalAs(UnmanagedType.ByValTStr, SizeConst = MAXPNAMELEN)]
            internal string szPname;
            internal Int16 wTechnology;
            internal Int16 wVoices;
            internal Int16 wNotes;
            internal Int16 wChannelMask;
            internal int dwSupport;
        }

        [StructLayout(LayoutKind.Sequential)]
        public struct MIDIINCAPS {
            internal Int16 wMid;
            internal Int16 wPid;
            internal int vDriverVersion;
            [MarshalAs(UnmanagedType.ByValTStr, SizeConst = MAXPNAMELEN)]
            internal string szPname;
        }
```

Ilustración 27. La clase win32. Definición de constantes y estructuras.

Para poder usar las funciones de Winmm.dll en C# es necesario traducir a este la definición de los tipos de datos y de las funciones de mmsystem.h.

En la Ilustración 27 y Ilustración 28 se muestra la clase win32 al completo.

En el caso del programa del proyecto se han utilizado las siguientes transformaciones de tipos en la clase win32[15]:

- Las constantes se definen mayoritariamente como enteros de 32 bits por lo tanto se han definido en C# con el prefijo **public** const **int**
- Los datos de tipo WORD en el API de Windows corresponden a un valor de 16 bits entero por lo tanto se tienen que transformar en C Sharp al tipo de dato **Int16.**
- Para los valores de tipo DWORD que es un tipo de 32 bits existe la correspondencia en C# como **Int.**
- Hay valores en el API de Windows del tipo TCHAR. Estos corresponden al tipo **string** en C#. Son de tipo cadena de caracteres.
- El tipo MMVERSION corresponde con uno de tipo UINT que es de 32 bits, por lo tanto la transformación en C# se realiza a **Int**.
- Para los datos estructurados es necesario especificar que mantienen un orden en memoria consecutivo. Hay que poner delante de la definición de la estructura el prefijo [StructLayout(**LayoutKind**.*Sequential*)].
- El tipo DWORD_PTR se utiliza para definir un puntero de tamaño 32 bits. En nuestro caso se usa como variable puntero a la función MidiInProc definida en el programa MIDIm por lo tanto la transformación requerida es a **Int.** Se usa una variable de tipo Int porque es un valor que pasamos nosotros a la función. Aunque no es un valor con el que poder hacer calculo matemáticos y si representa una dirección de memoria en donde se encontrará la función MidiInProc definida de usuario.
- Hay casos, como en el caso de recibir punteros, es decir, direcciones de memoria. Así que hacemos mención a una referencia recibida pues es un dato que hay que tratar por C# como puntero. Es el caso de, ver Ilustración 24, la función

midiInOpen del API de Windows en el que LPHMIDIIN corresponde con un tipo de puntero. Para este caso se hace referencia en la definición en C# como **ref IntPtr**, puntero referencia al dispositivo MIDI abierto. Este puntero es el que se pasa a la función MidiInClose para que se cierre la comunicación con ese dispositivo.

- En C# hay que especificar de qué fichero se importa la función mediante el prefijo [DllImport("winmm.dll")]. También hay que indicar **public** static extern antes del tipo devuelto por estas funciones para poder tener acceso a ellas en C#.

```
56          [DllImport("winmm.dll")]
57          public static extern Int16 midiOutGetNumDevs();
58
59          [DllImport("winmm.dll")]
60          public static extern int midiOutOpen(
61              ref IntPtr lphMidiOut,
62              int uDeviceID,
63              IntPtr dwCallback,
64              IntPtr dwInstance,
65              int dwFlags
66          );
67
68          [DllImport("winmm.dll")]
69          public static extern int midiOutShortMsg(
70              IntPtr hMidiOut,
71              int dwMsg
72          );
73
74          [DllImport("winmm.dll", EntryPoint = "midiOutGetDevCapsA")]
75          public static extern int midiOutGetDevCaps(
76              int uDeviceID,
77              ref MIDIOUTCAPS lpCaps,
78              int uSize
79          );
80
81          [DllImport("winmm.dll")]
82          public static extern int midiOutClose(IntPtr hMidiOut);
83
84          [DllImport("winmm.dll")]
85          public static extern int midiInClose(IntPtr hMidiIn);
86
87          [DllImport("winmm.dll", EntryPoint = "midiInGetDevCapsA")]
88          public static extern int midiInGetDevCaps(
89              int uDeviceID, ref MIDIINCAPS lpCaps,
90              int uSize
91          );
92
93          [DllImport("winmm.dll")]
94          public static extern int midiInGetNumDevs();
95
96          [DllImport("winmm.dll")]
97          public static extern int midiInOpen(
98              ref IntPtr lphMidiIn,
99              int uDeviceID,
100             MidiInProc dwCallback,
101             int dwInstance,
102             int dwFlags
103         );
104
105         [DllImport("winmm.dll")]
106         public static extern int midiInStart(IntPtr hMidiIn);
107     }
```

Ilustración 28. La clase win32. Definición de funciones o métodos.

Una vez definidas estas constantes, estructuras y funciones es necesario anteponer el prefijo win32 y un punto para hacer referencia a estas. Por ejemplo en el código se hace la definición de las variables que van a almacenar la dirección de memoria de la función que va a controlar las entradas de datos MIDI de esta manera: win32.MidiInProc *mipEnt1*, *mipEnt2*;

4.2.2 La clase Midiio

La clase Midiio es una clase instanciable que sirve para gestionar la entrada y salida MIDI. Y ofrece las siguientes características:

- Se produce una lectura de la configuración anterior de puertos usados y tipo inicial de funcionamiento almacenada en el fichero "MIDIm.sav".

- También se prepara la matriz que va a almacenar los datos de los controles MIDI desde la entrada y salida.

- Aquí se abren y cierran los dispositivos MIDI.

- Ofrece propiedades y métodos públicos que serán usados por la clase que instancia Midiio.

- Además, en esta clase se definen también las funciones callback que se encargarán de la lectura de datos del dispositivo MIDI.

La codificación de todas estas características se pueden comprobar más adelante en el código expuesto en la Ilustración 30.

A continuación se van a documentar las propiedades y métodos públicos que ofrece Midiio:

- void *iniciar*() lee la configuración de disco guardada e inicializa el valor anterior a un valor no posible (-1 por ejemplo)

- void *terminar*() guarda la configuración a disco y cierra todos los dispositivos abiertos

- string[] *obtenerEntradas*() obtiene un vector de "strings" con los nombres de los dispositivos de entrada cuyo índice los identificará en el programa.

- string[] *obtenerSalidas*() obtiene un vector de "strings" con los nombres de los dispositivos de entrada cuyo índice los identificará en el programa.

- Las propiedades *dispEntSec*, *dispEntControl*, *dispSalSec* permiten consultar el número asociado de dispositivo o establecerlo abriendo el dispositivo en ese momento. Los números asociados corresponden con el índice del array de string que resulta de *obtenerEntradas*() o *obtenerSalidas*() en su caso.

- La propiedad *dispTipo* permite conocer el tipo seleccionado o establecerlo. Los números asociados corresponden con el índice definido secuencial de 0 a 2 en *obtenerTipos*():

  string[] t = { "Normal", "Buscar", "Proporcional" };

- Las propiedades que almacenar el error: e*rrorAbrirDispEntSec*, *errorAbrirDispEntControl* y *errorAbrirDispSalSec*. Estas propiedades sirven para consultarlas en cualquier momento y representan el error de la última apertura. Se utilizan para mostrar una especie de indicadores "leds" de error en la ventana (verde si todo ha ido bien). Ver Ilustración 32.

Con estas propiedades y métodos públicos se accede de forma externa a las posibilidades de Midiio. Con respecto a los métodos y propiedades internas puede consultar el resto de código más adelante en la Ilustración 30.

Uno de los métodos más importantes corresponde a la definición de las funciones callback del tipo MidiInProc para cada uno de los dispositivos y por cada tipo de funcionamiento. Vamos a documentar estos métodos.

La estructura genérica de la función MidiInProc para todas las funciones de este tipo en el programa consiste en la mostrada en la Ilustración 29.

```
void procSec(int hMidiIn, int wMsg, int dwInstance, int dwParam1, int dwParam2)
{
    //recoge entrada del [redacted]
    byte[] datos;
    int estado, canal;
    if (wMsg == win32.MIM_DATA) { //si es un mensaje de canal
        datos = BitConverter.GetBytes(dwParam1);
        #if (DEBUG)
        Debug.Write(string.Format("Sec ->: {0:000}-{1:000}-{2:000}\n",
                                  datos[0], datos[1], datos[2]));
        #endif
        //obtener los 4 primeros bits del byte de estado para ver si es control: estado
        estado = datos[0] >> 4 & 0xF;
        //obtener los 4 siguientes bits del byte de estado: canal
        canal = datos[0] & 0xF;
        //datos[1] es el controlador y datos[2] el valor de este
        if (estado == CONTROLADOR_MIDI) {
            [redacted]
        }
    }
}
```

Ilustración 29. Esqueleto MidiInProc

Recordemos que la función se ejecuta cuando hay nuevos datos de entrada en el dispositivo al que se la ha asociado. En el código de la Ilustración 29 se puede ver que primero se comprueba si el mensaje es de tipo canal (solo

nos interesan los controles que afectan a cada canal. Como ya se ha visto, 128 por canal. A continuación y si esto es así, es un mensaje de canal, hay que averiguar si es control MIDI. Para ello se obtienen los datos, los 3 bytes, del mensaje MIDI con el parámetro *dwParam1*. El primer byte consiste en el estado y el canal. El estado tiene que ser en binario 1011 (B en hexadecimal) para que sea un "control change". Si es de tipo control se realizará el código que haya en el rectángulo azul. Ver Ilustración 19 y texto asociado.

En cada momento estarán a la espera 2 funciones de este tipo. La que permite recoger la entrada del secuenciador y la que permite recogerla del dispositivo controlador MIDI. En el segundo caso dependerá de la función del tipo de funcionamiento seleccionado para que sea una u otra. En total 4 funciones de tipo MidiInProc definidas: entrada del secuenciador, entrada del controlador pero procesando con tipo "normal", entrada del controlador pero procesando con tipo "buscar" y entrada del controlador pero procesando con tipo "proporcional". Se han definido 4 funciones MidiInProc por considerar que es mejor que el código esté todo secuencial en memoria y así no dependa de funciones o métodos llamados.

A continuación se presenta el código de la clase Midiio:

```
using System;
using System.Diagnostics;
using System.IO;
using System.Runtime.InteropServices;

namespace MIDIm
{
    class Midiio
    {
        //Constantes
        const int NUMERO_CONTROLADORES_MIDI = 128;
        const int NUMERO_CANALES = 16;
        const int VALOR_DEL_SECUENCIADOR = 0, VALOR_ANTERIOR = 1, VALOR_COMPROBAR = 2;
        const int COMPROBAR = 1, NO_COMPROBAR = 0;
        const int CONTROLADOR_MIDI = 0xB;

        //Variables
        win32.MIDIINCAPS mic = new win32.MIDIINCAPS();
        win32.MIDIOUTCAPS moc = new win32.MIDIOUTCAPS();
        win32.MidiInProc mipEntSec, mipEntControl;
        //variables puntero a funcion proceso mensajes midi
        IntPtr hmEntSec = IntPtr.Zero;
        IntPtr hmEntControl = IntPtr.Zero;
        IntPtr hmSalSec = IntPtr.Zero;
        //Numero identificativo de dispositivo
        int numDispEntSec, numDispEntControl, numDispSalSec, numDispTipo;
        public int errorAbrirDispEntSec;
        public int errorAbrirDispEntControl;
        public int errorAbrirDispSalSec;
        bool seAbrioDispEntSec = false;
        bool seAbrioDispEntControl = false;
        bool seAbrioDispSalSec = false;
        bool seSeleccionoTipo = false;

        //vector memoria: canal, controlador, valor (actual, anterior)
        Single[,,] memo;

        //Constructor
        public Midiio()
        {
            //16 canales, 128 controladores por canal, 3 valores a guardar: actual, anterior y comprobar
            memo = new Single[NUMERO_CANALES, NUMERO_CONTROLADORES_MIDI, 3];
            mipEntSec = procSec;
        }

```

```
        //Propiedades
        public int dispEntSec {
            get { return numDispEntSec; }
            set {
                numDispEntSec = value;
                if (seAbrioDispEntSec) {
                    cerrarDispEntSec();  //solo se cierra el puerto si se abrió previamente

                } else {
                    seAbrioDispEntSec = true;
                }
                abrirDispEntSec();
            }
        }
        public int dispEntControl {
            get { return numDispEntControl; }
            set {
                numDispEntControl = value;
                if (seAbrioDispEntControl) {
                    cerrarDispEntControl();  //solo se cierra el puerto si se abrió previamente

                } else {
                    seAbrioDispEntControl = true;
                }
                abrirDispEntControl();  //Abrir el dispositivo MIDI
            }
        }
        public int dispSalSec {
            get { return numDispSalSec; }
            set {
                numDispSalSec = value;
                if (seAbrioDispSalSec) {
                    cerrarDispSalSec();  //solo se cierra el puerto si se abrió previamente

                } else {
                    seAbrioDispSalSec = true;
                }
                abrirDispSalSec();
            }
        }
        public int dispTipo {
            get { return numDispTipo; }
            set {
                numDispTipo = value;
                if (seSeleccionoTipo) {
                    seleccionaTipo(numDispTipo);
                    cerrarTodosDisp();  //solo se cierra el puerto si se abrió previamente
                    abrirTodosDisp();
                } else {
                    seSeleccionoTipo = true;
                }
            }
        }

        //Metodos
        void procSec(int hMidiIn, int wMsg, int dwInstance, int dwParam1, int dwParam2)
        {
            //recoge entrada del secuenciador
            byte[] datos;
            int estado, canal;
            if (wMsg == win32.MIM_DATA) { //si es un mensaje de canal
                datos = BitConverter.GetBytes(dwParam1);
                #if (DEBUG)
                Debug.Write(string.Format("Sec ->: {0:000}-{1:000}-{2:000}\n",
                                          datos[0], datos[1], datos[2]));
                #endif
                //obtener los 4 primeros bits del byte de estado para ver si es control: estado
                estado = datos[0] >> 4 & 0xF;
                //obtener los 4 siguientes bits del byte de estado: canal
                canal = datos[0] & 0xF;
                //datos[1] es el controlador y datos[2] el valor de este
                if (estado == CONTROLADOR_MIDI) {
                    //Se guarda el valor del controlador que viene del secuenciador
                    memo[canal, datos[1], VALOR_DEL_SECUENCIADOR] = datos[2];
                    memo[canal, datos[1], VALOR_COMPROBAR] = COMPROBAR; //para tipo buscar
                }
            }
        }
```

```
124         void procCont_normal(int hMidiIn, int wMsg, int dwInstance, int dwParam1, int dwParam2)
125         {
126             //recoge entrada del controlador, la cambia y la devuelve al secuenciador
127             byte[] datos;
128             int estado, canal;
129             if (wMsg == win32.MIM_DATA) { //si es un mensaje de canal
130                 datos = BitConverter.GetBytes(dwParam1);
131                 #if (DEBUG)
132                 Debug.Write(string.Format("ctr ->: {0:000}-{1:000}-{2:000}",
133                                           datos[0], datos[1], datos[2]));
134                 #endif
135                 //obtener los 4 primeros bits del byte de estado para ver si es control: estado
136                 estado = datos[0] >> 4 & 0xF;
137                 //obtener los 4 siguientes bits del byte de estado: canal
138                 canal = datos[0] & 0xF;
139                 //datos[1] es el controlador y datos[2] el valor de este
140                 if (estado == CONTROLADOR_MIDI) {
141                     memo[canal, datos[1], VALOR_ANTERIOR] = datos[2];
142                     enviar(datos);
143                 }
144             }
145         }
146         void procCont_buscar(int hMidiIn, int wMsg, int dwInstance, int dwParam1, int dwParam2)
147         {
148             //recoge entrada del controlador, la cambia y la devuelve al secuenciador
149             byte[] datos;
150             int estado, canal;
151             if (wMsg == win32.MIM_DATA) { //si es un mensaje de canal
152                 datos = BitConverter.GetBytes(dwParam1);
153                 #if (DEBUG)
154                 Debug.Write(string.Format("ctr ->: {0:000}-{1:000}-{2:000}",
155                                           datos[0], datos[1], datos[2]));
156                 #endif
157                 //obtener los 4 primeros bits del byte de estado para ver si es control: estado
158                 estado = datos[0] >> 4 & 0xF;
159                 //obtener los 4 siguientes bits del byte de estado: canal
160                 canal = datos[0] & 0xF;
161                 //datos[1] es el controlador y datos[2] el valor de este
162                 if (estado == CONTROLADOR_MIDI) {
163                     // disable once CompareOfFloatsByEqualityOperator
164                     if (memo[canal, datos[1], VALOR_COMPROBAR] == NO_COMPROBAR) {
165                         enviar(datos);
166                     } else {//comprobar primero si se ha llegado con el controlador al valor actual
167                         if (memo[canal, datos[1], VALOR_DEL_SECUENCIADOR] > datos[2] - 3
168                             && memo[canal, datos[1], VALOR_DEL_SECUENCIADOR] < datos[2] + 3) {
169                             memo[canal, datos[1], VALOR_COMPROBAR] = NO_COMPROBAR;
170                             Console.Beep();
171                         }
172                     }
173                     memo[canal, datos[1], VALOR_ANTERIOR] = datos[2];
174                 }
175             }
176         }
```

```
177         void procCont_proporcional(int hMidiIn, int wMsg, int dwInstance, int dwParam1, int dwParam2)
178         {
179             //recoge entrada del controlador, la cambia y la devuelve al secuenciador
180             byte[] datos;
181             int estado, canal;
182             Single aux;
183             Single dato2, vAnt, vSec;
184             if (wMsg == win32.MIM_DATA) { //si es un mensaje de canal
185                 datos = BitConverter.GetBytes(dwParam1);
186                 #if (DEBUG)
187                 Debug.Write(string.Format("ctr ->: {0:000}-{1:000}-{2:000}",
188                                           datos[0], datos[1], datos[2]));
189                 #endif
190                 //obtener los 4 primeros bits del byte de estado para ver si es control: estado
191                 estado = datos[0] >> 4 & 0xF;
192                 //obtener los 4 siguientes bits del byte de estado: canal
193                 canal = datos[0] & 0xF;
194                 //datos[1] es el controlador y datos[2] el valor de este
195                 if (estado == CONTROLADOR_MIDI) { //si es controlador midi
196                     // disable once CompareOfFloatsByEqualityOperator
197                     if (memo[canal, datos[1], VALOR_ANTERIOR] == -1) { //si no se ha cambiado nunca
198                         memo[canal, datos[1], VALOR_ANTERIOR] = datos[2];
199                         memo[canal, datos[1], VALOR_DEL_SECUENCIADOR] = datos[2];
200                     } else {
201                         // en este caso habrá un buen valor anterior para calcular la proporción
202                         dato2 = datos[2];
203                         aux = dato2;
204                         vAnt = memo[canal, datos[1], VALOR_ANTERIOR];
205                         vSec = memo[canal, datos[1], VALOR_DEL_SECUENCIADOR];
206                         if (memo[canal, datos[1], VALOR_ANTERIOR] < dato2) {
207                             dato2 = (dato2 - vAnt) * (127 - vSec) / (127 - vAnt) + vSec;
208                         } else {
209                             dato2 = vSec * (vAnt + (dato2 - vAnt)) / vAnt;
210                         }
211                         //se devuelve al array de bytes para enviarse al secuenciador
212                         datos[2] = (byte)dato2;
213                         //valor calculado proporcional
214                         memo[canal, datos[1], VALOR_DEL_SECUENCIADOR] = dato2;
215                         //valor anterior del controlador
216                         memo[canal, datos[1], VALOR_ANTERIOR] = aux;
217                         enviar(datos);
218                     }
219                 }
220             }
221         }
222         void enviar(byte[] datos)
223         {
224             int canal = datos[0] & 0xF;
225             #if (DEBUG)
226             Debug.Write(string.Format(" {0:000} -> Sec\n", datos[2]));
227             #endif
228             int dato = BitConverter.ToInt32(datos, 0);
229             if (hmSalSec != IntPtr.Zero)
230                 win32.midiOutShortMsg(hmSalSec, dato);
231         }
```

```
232         public string[] obtenerEntradas()
233         {
234             int numDevs = win32.midiInGetNumDevs();
235             string[] s = new string[numDevs];
236             for (int i = 0; i < numDevs; i++) {
237                 win32.midiInGetDevCaps(i, ref mic, Marshal.SizeOf(mic));
238                 s[i] = mic.szPname;
239             }
240             return s;
241         }
242         public string[] obtenerSalidas()
243         {
244             int numDevs = win32.midiOutGetNumDevs();
245             string[] s = new string[numDevs];
246             for (int i = 0; i < numDevs; i++) {
247                 win32.midiOutGetDevCaps(i, ref moc, Marshal.SizeOf(moc));
248                 s[i] = moc.szPname;
249             }
250             return s;
251         }
252         public string[] obtenerTipos()
253         {
254             string[] t = { "Normal", "Buscar", "Proporcional" };
255             return t;
256         }
257         void abrirDispEntSec()
258         {
259             hmEntSec = IntPtr.Zero;
260             errorAbrirDispEntSec = win32.midiInOpen(
261                 ref hmEntSec, dispEntSec, mipEntSec, 0, win32.CALLBACK_FUNCTION);
262             win32.midiInStart(hmEntSec);
263         }
264         void abrirDispEntControl()
265         {
266             hmEntControl = IntPtr.Zero;
267             errorAbrirDispEntControl = win32.midiInOpen(
268                 ref hmEntControl, dispEntControl, mipEntControl, 0, win32.CALLBACK_FUNCTION);
269             if (hmEntControl != IntPtr.Zero)
270                 win32.midiInStart(hmEntControl);
271         }
272         void abrirDispSalSec()
273         {
274             hmSalSec = IntPtr.Zero;
275             errorAbrirDispSalSec = win32.midiOutOpen(
276                 ref hmSalSec, dispSalSec, IntPtr.Zero, IntPtr.Zero, 0);
277         }
278         public void iniciar()
279         {
280             leerConfig();
281             iniciarValorAnterior(-1);
282         }
283         void leerConfig()
284         {
285             try {
286                 FileStream fichero = new FileStream("MIDIMemo.sav", FileMode.Open);
287                 dispEntSec = fichero.ReadByte();
288                 dispEntControl = fichero.ReadByte();
289                 dispSalSec = fichero.ReadByte();
290                 dispTipo = fichero.ReadByte();
291                 fichero.Close();
292             } catch (Exception) { // en el caso de haber algun cambio de dispositivos MIDI
293                 dispEntSec = 0;
294                 dispEntControl = 0;
295                 dispSalSec = 0;
296                 dispTipo = 0;
297             }
298             seleccionaTipo(dispTipo);
299         }
300         void guardarConfig()
301         {
302             try {
303                 FileStream fichero = new FileStream("MIDIMemo.sav", FileMode.Create);
304                 fichero.WriteByte(Convert.ToByte(dispEntSec));
305                 fichero.WriteByte(Convert.ToByte(dispEntControl));
306                 fichero.WriteByte(Convert.ToByte(dispSalSec));
307                 fichero.WriteByte(Convert.ToByte(dispTipo));
308                 fichero.Close();
309             } catch (Exception) {
310                 Console.WriteLine("Error al grabar MIDIMemo.sav");
311             }
312         }
```

```
313         void cerrarTodosDisp()
314         {
315             cerrarDispEntSec();
316             cerrarDispEntControl();
317             cerrarDispSalSec();
318         }
319         void abrirTodosDisp()
320         {
321             abrirDispEntSec();
322             abrirDispEntControl();
323             abrirDispSalSec();
324         }
325         void cerrarDispEntSec()
326         {
327             win32.midiInClose(hmEntSec);
328         }
329         void cerrarDispEntControl()
330         {
331             win32.midiInClose(hmEntControl);
332         }
333         void cerrarDispSalSec()
334         {
335             win32.midiOutClose(hmSalSec);
336         }
337         public void terminar()
338         {
339             guardarConfig();
340             cerrarTodosDisp();
341         }
342         void iniciarValorAnterior(int valor)
343         {
344             for (int canal = 0; canal < NUMERO_CANALES; canal++) {
345                 for (int control = 0; control < NUMERO_CONTROLADORES_MIDI; control++)
346                     memo[canal, control, VALOR_ANTERIOR] = valor;
347             }
348         }
349         void seleccionaTipo(int tipo)
350         {
351             switch (tipo) {
352                 case 0: //normal
353                     mipEntControl = procCont_normal;
354                     break;
355                 case 1: //buscar
356                     mipEntControl = procCont_buscar;
357                     break;
358                 case 2: //proporcional
359                     mipEntControl = procCont_proporcional;
360                     break;
361                 default:
362                     mipEntControl = procCont_normal;
363                     break;
364             }
365         }
366     }
367 }
368
```

Ilustración 30. Clase Midiio.

Las 4 funciones o métodos del tipo MidiInProc son: *procSec*, *procControl_normal*, *procControl_buscar* y *procControl_proporcional*.

La función *procSec*, en la parte cuando ya se sabe que se ha obtenido un "control change", se dedica a guardar el valor que viene del secuenciador para el control del canal determinado en la columna de la matriz VALOR_DEL_SECUENCIADOR y también a almacenar COMPROBAR en la columna VALOR_COMPROBAR para que se compruebe si se ha procesado algún valor anteriormente en el caso de seleccionar el tipo "buscar".

La función *procControl_normal* se comporta de la misma forma que si uniésemos el controlador MIDI con el secuenciador sin pasar por el programa

MIDIm. Almacena el valor en la columna VALOR_ANTERIOR de la matriz y envía de nuevo los datos sin modificar al secuenciador.

En la función *procControl_buscar* se comprueba si el valor del control leído está cerca del valor en el secuenciador almacenado. No se envía nada hasta que está dentro del intervalo no cambiando el valor en el secuenciador hasta que el que movemos desde el controlador se ajuste a un intervalo dentro del cual está el valor del secuenciador. Parece que el valor se busca del controlador MIDI hasta que se encuentra en un intervalo

Por último, la función *procControl_proporcional* comprueba la dirección hacia donde estamos moviendo el controlador pues se mira el valor anterior y se compara con el leído. Después se calcula el valor proporcional que queda hasta el principio o hasta el final del potenciómetro según sea el caso

El resto de métodos o funciones y propiedades pueden ser comprobados en el código de la Ilustración 30.

4.3 Ejecución

El programa en primera instancia aparece minimizado, cuando se ejecuta por primera vez o cuando se minimiza de nuevo, solo con el icono en el área de notificaciones de Windows. Ver Ilustración 31.

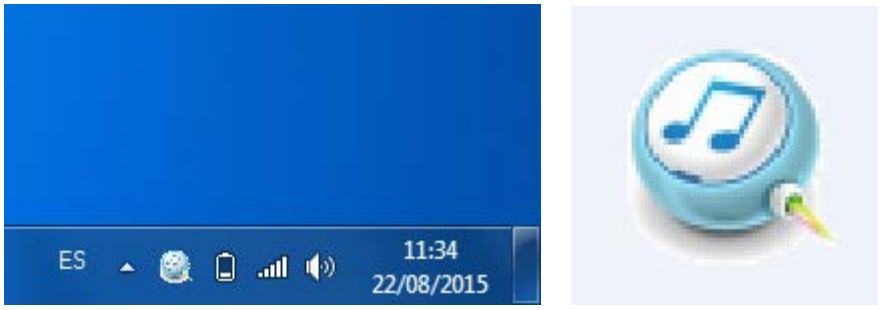


Ilustración 31. Área de notificaciones de Windows e icono.

En la fase de diseño se ha definido la ventana Windows del programa y los controles Windows necesarios para que tenga el aspecto de la Ilustración 32.

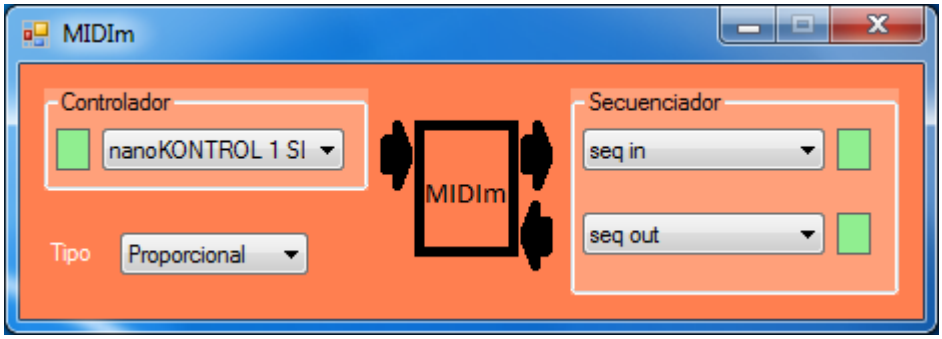


Ilustración 32. Ventana del programa MIDIm.

Como puede observarse se ha definido la entrada del controlador MIDI, y la entrada y salida del secuenciador con desplegables con la indicación en color verde o rojo si se abierto correctamente o no, respectivamente. También está definido otro desplegable para poder seleccionar el tipo de funcionamiento a usar.

La apertura de puertos e inicialización se produce cada vez que cambiamos el dispositivo en los desplegables mostrando en ese momento si hay error de apertura en el “led” cercano.

4.4 Vídeo demostrativo

Para demostrar de forma gráfica las capacidades del programa desarrollado se ha grabado un vídeo en el que se presenta el funcionamiento del programa haciendo hincapié en los tipos de funcionamiento. El vídeo se puede encontrar en la dirección https://youtu.be/VBXqmerQL9s y además se visualizará en la defensa del trabajo.

5 Conclusiones y trabajo futuro

5.1 Conclusiones

Este proyecto ha pretendido dar soluciones a un problema con controladores MIDI que no permiten representar el valor de control de vuelta. Estos no disponen de entrada pues no son capaces de representar estos valores.

La solución pasa por memorizarlos aparte y dar distintos tipos de soluciones.

El programa funciona correctamente haciendo su cometido y solucionando el problema de una forma elegante para los medios disponibles.

La solución óptima para el problema que se plantea son los controladores motorizados, pero que mi proyecto permite solucionarlo de forma satisfactoria con una importante reducción de costes.

5.2 Trabajo futuro

Se ha dejado para una mejora del proyecto la posibilidad de seleccionar el intervalo en el caso del tipo buscar.

También se podría mejorar haciendo posible configurar individualmente el tipo de funcionamiento. Es decir, para cada uno de los controles.

Se podría mejorar el rendimiento utilizando un lenguaje como C++. En algunos momentos se nota cierta lentitud en la ejecución aunque no afecta a la experiencia de usuario.

También sería importante no tener que depender de puertos virtuales MIDI mediante el programa LoopMIDI y que el programa MIDIm los creara por sí solo. Así se evitarían alguno de los requisitos de funcionamiento.

6 Bibliografía

[1] Steinberg. ..... www.steinberg.net
acceso: 10/05/2015

[2] Korg. ..... www.korg.com
acceso: 10/05/2015

[3] Novation. ..... us.novationmusic.com
acceso: 10/05/2015

[4] Akai. ..... www.akaipro.com
acceso: 10/05/2015

[5] Roland. ..... es.wikipedia.org
acceso: 14/05/2015

[6] ReBirth..... www.propellerheads.se
acceso: 07/08/2015

[7] Conectores MIDI. ..... es.wikipedia.org
acceso: 07/08/2015

[8] LoopMIDI ..... www.tobias-erichsen.de
acceso: 07/08/2015

[9] Reaper ..... www.reaper.fm
acceso: 07/08/2015

[10] Swift, Andrew. (May 1997), "A brief Introduction to MIDI", SURPRISE (Imperial College of Science Technology and Medicine)

[11] CSS audiovisual ..... www.css-audiovisual.com
acceso: 10/08/2015

[12] DX1 technical specifications ..... www.kratzer.at
acceso: 10/08/2015

[13] MSDN..... msdn.microsoft.com
acceso: 20/08/2015

[14] Sharpdevelop..... www.icsharpcode.net
acceso: 13/08/2015

[15] StackOverflow ..... http://stackoverflow.com/
acceso: 29/08/2015