COMPILADOR PARA LA GENERACIÓN DE MÚSICA EN FORMATO MIDI

Fernando Chávez[1], Joaquin Castelo[2], Marcelo Quina[3], Roxana Flores[4].
Universidad Nacional de San Agustín
[1]fchavezme@unsa.edu.pe, [2]mquinad@unsa.edu.pe, [3]jcasteloc@unsa.edu.pe, [4]rfloresqu@unsa.edu.pe

Abstract. Given the growing interest in algorithmic composition and music generation, the need for flexible and programmable tools to create music has increased. This article presents a compiler designed to generate music in MIDI format using a custom high-level language tailored for musical structures. The compiler processes input code written in this language, applies a defined grammar and semantics, and translates it into standard MIDI instructions. The system allows composers and developers to define melodies, harmonies, rhythms, and dynamics with precision and structure. Experimental tests show that the compiler successfully produces coherent and expressive musical pieces, demonstrating its effectiveness for automated composition and educational purposes.

Keywords. music compiler, MIDI generation, algorithmic composition, music programming.

1. INTRODUCCIÓN

El presente proyecto tiene como objetivo principal el diseño e implementación de un compilador especializado en la generación de archivos MIDI (Musical Instrument Digital Interface) a partir de un lenguaje propio de descripción musical. Este compilador permitirá a los usuarios escribir composiciones musicales utilizando una sintaxis estructurada, similar a la de un lenguaje de programación, y traducir dichas instrucciones en composiciones musicales reproducibles mediante cualquier software compatible con el formato MIDI.

La problemática que aborda este proyecto radica en la dificultad que tienen muchas personas que desean crear música, pero no cuentan con conocimientos musicales formales o habilidades para tocar un instrumento. Por ejemplo, en un piano físico, no siempre es evidente qué nota corresponde a cada tecla, lo que puede limitar la exploración musical para quienes no dominan la teoría o la práctica instrumental. Con este compilador, no será necesario saber leer partituras ni identificar las notas en un instrumento real; bastará con escribir instrucciones textuales simples para componer piezas completas, facilitando el acceso a la creación musical mediante la programación.

El enfoque del proyecto combina conceptos de teoría de lenguajes formales, análisis léxico y sintáctico, estructuras de datos y teoría musical básica, permitiendo explorar de manera interdisciplinaria la relación entre programación y música. El lenguaje propuesto incluirá instrucciones para representar notas, duraciones, silencios, acordes, compases y otros elementos fundamentales de una partitura musical.

Para llevar a cabo este proyecto, se diseñará un lenguaje de alto nivel enfocado exclusivamente en la composición musical textual. Posteriormente, se construirá un compilador que recorrerá distintas etapas: análisis léxico, para identificar los componentes del lenguaje; análisis sintáctico, para validar la estructura de las composiciones; y generación de código, donde se traducirá la representación abstracta de la música en un archivo .mid ejecutable en cualquier secuenciador MIDI o programa de edición musical digital.

El desarrollo del sistema se dividirá en varias etapas: diseño del lenguaje, implementación del analizador léxico y sintáctico, generación del archivo MIDI, y finalmente validación mediante casos de prueba con diferentes piezas musicales.

Se espera que este proyecto no solo facilite la creación musical para personas con conocimientos en programación, sino que también contribuya al campo de la composición algorítmica, ofreciendo una herramienta didáctica y funcional que demuestre cómo la música puede ser concebida, escrita y reproducida mediante código.

2. ESTADO DEL ARTE

El desarrollo de un compilador musical que traduzca una sintaxis simbólica en archivos de audio en formato MIDI se fundamenta en diversos enfoques previamente explorados en la composición algorítmica, la programación musical y la generación sonora automatizada. El uso de gramáticas formales, lenguajes específicos y abstracciones musicales ha demostrado ser efectivo para estructurar el proceso de creación musical desde la computación.

En el estudio "Six Techniques for Algorithmic Music Composition", Langston [5] presenta una serie de algoritmos orientados a la generación automática de música, entre ellos el uso de L-systems, que permiten definir reglas sintácticas generativas para construir estructuras musicales auto-similares. Este enfoque resulta particularmente relevante para un compilador musical, ya que ofrece un modelo formal capaz de producir salidas coherentes en formato MIDI a partir de entradas simbólicas.

Por otro lado, Loy y Abbott [4] abordan el uso de lenguajes formales en la representación, síntesis y composición musical mediante computadora. En su trabajo se analizan paradigmas de lenguajes musicales como los descriptivos, interpretativos y generativos, permitiendo modelar estructuras musicales complejas y traducirlas en instrucciones ejecutables. Esta

visión refuerza la idea de un compilador como intermediario entre la notación musical abstracta y su ejecución digital.

Ince [3], en su investigación sobre abstracción en programación musical, introduce el sistema *Siren*, una interfaz para composición basada en patrones textuales y funciones musicales. Su arquitectura modular y la capacidad para manipular expresiones simbólicas lo convierten en una herramienta aplicable como lenguaje intermedio dentro de un compilador musical, permitiendo transformar código en eventos sonoros dentro de entornos como SuperCollider.

Asimismo, el lenguaje propuesto por De Paz et al. [1] permite la composición automática de música tonal mediante operadores de alto nivel que integran modelos estocásticos basados en cadenas de Markov. El diseño imperativo de este lenguaje y su capacidad para generar múltiples salidas coherentes a partir de un mismo código fuente se alinea con los objetivos de un compilador musical, al facilitar la generación estructurada de obras musicales reproducibles en formato MIDI.

Finalmente, Anderson y Bilmes [2] desarrollan MOOD, un sistema basado en C++ para la generación de música en tiempo real mediante procesos concurrentes. Su diseño jerárquico, que abarca planificación temporal, estructuras virtuales de tiempo y abstracciones musicales, permite que el código fuente se traduzca directamente en eventos musicales. Esto representa una implementación concreta de un compilador que integra análisis, interpretación y ejecución de manera simultánea.

Estos antecedentes proporcionan un marco técnico sólido para el desarrollo de compiladores musicales, demostrando la viabilidad de transformar lenguajes simbólicos de alto nivel en representaciones musicales digitales de forma estructurada, expresiva y automatizada.

3. METODOLOGÍA

A. FUNCIONAMIENTO DEL COMPILADOR

El compilador propuesto tiene como objetivo transformar un archivo de texto que contiene instrucciones musicales a un archivo de salida en formato MIDI para que este pueda ser interpretado por cualquier software de reproducción musical. Para lograr este objetivo diseñamos un pipeline de procesamiento por etapas.

El proceso completo se divide en siete etapas consecutivas, donde cada una cumple un rol específico en la traducción y generación de la música. Estas etapas permiten un procesamiento modular y sistemático de la información de entrada, garantizando la correctitud sintáctica, la detección de patrones musicales y una asignación estructurada de instrumentos.

Para ilustrar su funcionamiento se presenta la figura 1.

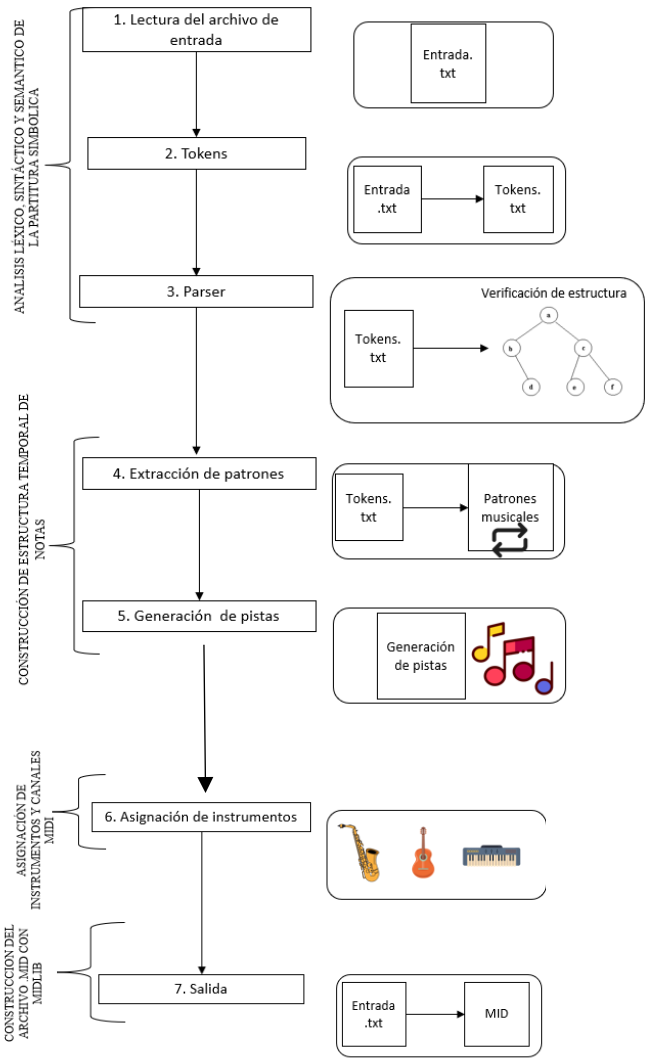


Fig. 1. Pipeline

Para convertir nuestro archivo de entrada en un archivo de salida .mid nuestro proceso se divide en diferentes etapas, cada una de estas cumple diferentes funciones específicas, a continuación, se describirá cada una de ellas de manera más detallada.

3.1. ANALISIS LÉXICO, SINTÁCTICO Y SEMANTICO DE LA PARTITURA SIMBOLICA

En esta primera etapa se realizan los análisis del archivo de texto de entrada. Estos se procesan línea por línea por símbolos musicales definidos por el usuario, también se definen los análisis necesarios para el compilador.

3.1.1 ANALISIS LÉXICO

En esta etapa se recorre línea por línea el archivo de entrada, para identificar las unidades léxicas o tokens. Estos tokens representan los componentes básicos del lenguaje, como notas musicales, compases, octavas, repetidores, bloques, silencios, y decoradores.

Los principales son:

-Notas Musicales:

Están definidas de la siguiente forma (nota, figura), el primer elemento representa el nombre de la nota y el segundo elemento la duración rítmica usando abreviaturas las usadas son:

Símbolo	Figura
r	Redonda
b	Blanca
c	corchea
f	Fusa
sc	Semicorchea
sf	semifusa

Estas figuras permiten representar la duración relativa de cada nota dentro del compás, siendo interpretadas como valores numéricos durante el análisis semántico.

-Silencios:

Se representan de la siguiente forma (sile, figura) donde sile nos indica que se trata de un silencio y figura que indica la duración del silencio con la misma simbología de las notas.

Cada figura musical tiene su correspondiente silencio, lo que permite representar compases con pausas completas o parciales. Aunque los silencios no egeneren sonido en el archivo MIDI, si afectan a la línea temporal de reproducción, lo que ayuda a desplazar la entrada de notas posteriores.

-Instrucciones de compás

Son expresiones como compas 4/4, estas definen la métrica de la canción. Aunque el archivo MIDI no posea una métrica audible, esta información ayuda internamente para validar la estructura rítmica de los bloques musicales.

-Cambios de Octavas

Son instrucciones de la forma oct=4 estas permiten establecer la octava activa, toda nota definida posteriormente se interpretará en esta octava, lo que afectará directamente el valor del pitch MIDI. Por ejemplo, do en la octava 4 sera la nota 60 en la escala de MIDI. Al cambiar la octava de altera la tesitura general de las pistas generadas.

-Definiciones de patrones

Se presentan con la sintaxis ritmo1 {}, aquí se define el contenido del bloque entre llaves, aquí se grupa una secuencia de notas o silencios, lo que permite su reutilización en distintas partes de la composición. Estas definiciones se almacenan como plantillas que pueden expandirse cuando se invoquen con repeticiones.

-Repeticiones

Son instrucciones como repeat(2) ritmo 1 permiten repetir un patrón previamente definido una cierta cantidad de veces. Esto facilita la construcción de estructuras musicales con secciones repetitivas. Cada repetición se procesa como una expansión temporal del patrón y se convierte en una pista independiente en el archivo MIDI.

-Asignación de instrumentos

La notación instrumento ritmo1 = 25, con esto se asigna un timbre musical al patron. El número corresponde al estándar general MIDI, por ejemplo, 25 representa la guitarra acústica. Al compilar, este valor se traduce gracias a un comando MIDI Program Change, que indica que instrumento utilizar para la pista correspondiente en la reproducción final.

Cada una de estas líneas se segmenta en tokens usando expresiones regulares, generando una tabla intermedia que especifica el tipo de cada símbolo reconocido. Lo que garantiza que el lenguaje definido sea estructurado, y que cada elemento pueda posteriormente traducirse a información musical digital.

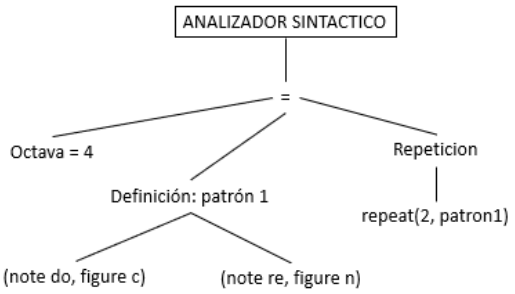
Mas adelante un analizador sintáctico verifica que las combinaciones de tokens respeten la gramática del lenguaje musical. Solo si la estructura es válida se procede a construir un modelo abstracto interno que representa las frases musicales en términos de compases, notas, duraciones y patrones reutilizables.

3.1.2 ANALISIS SINTACTICO

Se validan las secuencias de tokens obtenida en el análisis léxico, se verifica que cumpla las reglas gramaticales del lenguaje musical definido.

A través de funciones como parse(), bloque(), y compasRitmico(), se verifica el orden correcto de elementos como bloques, compases, notas y repeticiones. Si la secuencia no se ajusta a la sintaxis esperada, se lanza un error indicando la línea y el tipo de problema.

Árbol sintáctico



3.1.3 ANALISIS SEMANTICO

Durante esta fase se validan las relaciones lógicas y el significado musical de los elementos, se asegura que los patrones definidos existan cuando son involucrados, las octavas estén correctamente asignadas y dentro de un rango válido, además de que las notas tengan una duración

y pitch válidos, y finalmente que las repeticiones estén correctamente estructuradas.

Además, esta etapa prepara los datos para generar eventos MIDI: cada nota se convierte en una tupla (tiempo, duración, pitch), que luego se inserta pista por pista en el archivo .mid.

Algorithm 1 Etapa 1 - Análisis Léxico

```
for cada línea en entrada.txt do
    Eliminar espacios al inicio
    while la línea no esté vacía do
        if se encuentra coincidencia con algún patrón then
            Generar token (tipo, lexema, línea)
            Avanzar al siguiente segmento
        else
            Generar token tipo Desconocido
        end if
    end while
end for
Guardar tokens en tokens.txt
```

Algorithm 2 Etapa 1 - Análisis Sintáctico y Semántico

```
while haya tokens do
    if es instrucción de configuración (octava, compás, etc.) then
        Actualizar variables internas
    else if es definición de patrón then
        Almacenar tokens del bloque en patronesDefinidos[nombre]
    else if es instrucción repeat then
        Ejecutar subparser para repetir patrón y acumular notas
    end if
end while
```

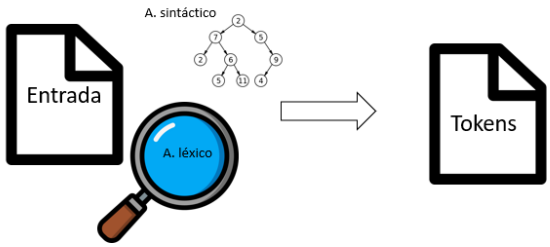


Fig. 2. Representation Etapa 1

3.2. CONSTRUCCIÓN DE ESTRUCTURA TEMPORAL DE NOTAS

Una vez validada la gramática y estructura del lenguaje simbólico musical en las etapas anteriores, el compilador procede a construir una representación interna de la música basada en eventos temporales. Esta capa es muy importante, ya que define cómo las instrucciones escritas por el usuario se convierten en datos precisos que representan el tiempo, la altura (pitch) y la duración de cada nota, fundamentales para su correcta conversión a formato MIDI.

Cada nota o silencio valido se transforma en una tupla conformada por tres componentes:

-Tiempo de inicio: Aquí se define el momento exacto en que la nota comienza a sonar.

-Duración: Esto se traduce directamente desde la figura musical que se coloca en el archivo de entrada.

-Pitch MIDI: Aquí se coloca el numero entero que corresponde a la nota según el estándar MIDI, por ejemplo, D04 = 60, RE#4 = 63, etc.

Estas tuplas se almacenan secuencialmente y se organizan cronológicamente para cada pista musical.

El tiempo de inicio no es explicito en el código fuente, se calcula acumulativamente, es decir, el compilador mantiene una variable interna de tiempo que se incrementa tras cada evento (nota o silencio). Por ejemplo, si una nota tiene duración 1.0 (una negra), la siguiente empezará automáticamente en el tiempo 1.0, salvo que haya silencios o cambios de figura intermedios.

Asimismo, los puntillos (representados con “.”) se detectan y afectan la duración final de una nota, multiplicándola por 1.5. Esta lógica se aplica antes de añadir el evento a la estructura temporal.

Los silencios a pesar de no generar sonido influyen directamente en la ubicación de notas posteriores, Cuando se detecta un evento como (sile,n), el compilador no agrega una nota a la estructura, pero incrementa el contador de tiempo, garantizando que la siguiente nota comience donde corresponde.

Esto mantiene la sincronía exacta del ritmo y permite que MidLib reciba una secuencia temporal coherente.

Para la reutilización de bloques musicales, cuando se procesa la instrucción “repeat(n) ritmo 1”, el compilador expande el contenido del patrón n veces y lo concatena de forma continua respetando el tiempo actual.

Cada patrón expandido se convierte en una pista independiente, lo que permite que MidLib genere múltiples canales en paralelo (cada uno con su propio instrumento si está definido).

Al finalizar esta etapa, el compilador posee una estructura de datos organizada por pistas, cada una de estas pistas contiene una lista ordenada de eventos (tiempo, duración, pitch) completamente lista para ser transformada a el resultado final, un archivo de formato MIDI. Aquí es donde Midlib entra en acción, permitiendo tomar cada una de estas pistas y construyendo un objeto musical, sus principales pasos son:

-Asigna a cada pista un canal MIDI distinto.

-Inserta las notas en orden, respetando sus tiempos y duraciones.

-Convierte cada tupla en eventos Note On y Note Off con temporización relativa.

-Añade información de instrumento por canal si fue definida en capas anteriores.

El uso de MidLib simplifica la conversión porque abstrae los detalles técnicos del formato binario MIDI (como codificación de eventos, estructura de encabezados, resolución temporal, etc.) y permite enfocarse únicamente en los aspectos musicales.

Algorithm 3 Etapa 3 - Procesamiento de Notas y Silencios

```
for cada nota o silencio en el patrón do
    if es silencio then
        tiempoActual ← tiempoActual + duración
    else if es nota then
        pitch ← convertirNotaAMIDI(nombreNota, octava)
        Agregar (tiempoActual, duración, pitch) a notas
        tiempoActual ← tiempoActual + duración
    end if
    if hay puntillo then
        duración ← duración × 1,5
    end if
end for
```

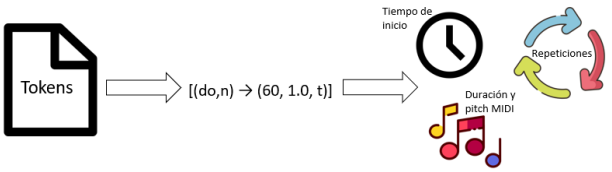


Fig. 3. Representation Etapa 2

3.3. ASIGNACIÓN DE INSTRUMENTOS Y CANALES MIDI

Una vez que las estructuras temporales de notas ya han sido generadas y organizadas en diferentes pistas, el compilador procederá a definir el timbre de cada una de las pistas, es decir, el instrumento que sonara en la reproducción de cada línea musical.

Esta etapa es muy importante para dotar a la composición de una riqueza tímbrica realista y para aprovechar el potencial polifónico del formato MIDI.

Para detectar las asignaciones musicales, el compilador permite al usuario definir instrumentos usando la siguiente sintaxis "instrumento ritmo1 = 25", donde ritmo1 es el patrón o pista que ya se ha definió antes y 25 es el número de instrumento según la especificación general MIDI, donde cada número corresponde a un timbre, por ejemplo 0 = piano acústico, 24 = guitarra acústica, etc.

Durante esta etapa el compilador recorre los tokens y extrae todas las asignaciones de instrumentos, construyendo un mapa que asocia cada nombre de pista con su instrumento respectivo.

En esta etapa también tenemos la lógica de asociación entre pistas y canales de MIDI, aquí el estándar MIDI permite un máximo de 16 cabales simultáneos los cuales están numerados de 0 al 15, donde cada uno de estos canales puede tener un instrumento distinto, con esto el compilador usa esta capacidad para generar música polifónica asignando:

-Una pista musical independiente para cada patrón repetido,

-Un canal MIDI exclusivo a cada pista (evitando colisiones entre instrumentos),

-El instrumento correspondiente según la definición del

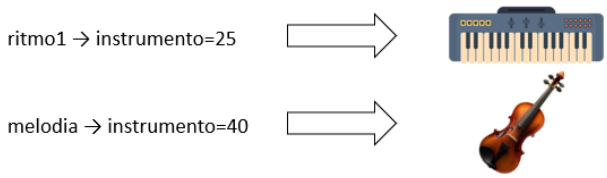


Fig. 4. Representation Etapa 3

3.4. CONSTRUCCION DEL ARCHIVO .MID CON MIDLIB

Una vez ya se organizaron todas las pistas musicales, se hayan asignado instrumentos a cada una de estas y definido todos los eventos musicales con precisión temporal, el compilador comienza con la etapa final. Para esta tarea se emplea la biblioteca MidLib, especializada en la generación de archivos MIDI desde C++.

Las estructuras generadas hasta este punto se encuentran organizadas como una colección de pistas musicales, cada una de estas posee:

-Tiempo de inicio (en beats),

-Duración de cada nota,

-Pitch MIDI,

-Canal e instrumento asignado previamente.

Esta información se encuentra en las estructuras internas del compilador, lo que sigue es traducir estas estructuras a un formato que cumpla con MIDI tipo 1, el cual puede soportar múltiples pistas con sus propios canales y eventos.

Posteriormente para la creación del objeto "Sample" la biblioteca MidLib ofrece un objeto central llamado "Sample" el cual representa una composición musical

completa en memoria. A este objeto se le agregan pistas (tracks), y cada pista puede contener múltiples eventos musicales.

El compilador crea el objeto con tantas pistas como líneas musicales haya definido el usuario. Cada pista es asociada a un canal MIDI único. Luego, se realiza lo siguiente:

-Se selecciona la pista activa dentro del objeto Sample.

-Se asigna el instrumento utilizando un evento MIDI Program Change, a través del método de MidLib.

-Se insertan todas las notas correspondientes, utilizando eventos note, especificando:

-Tiempo de inicio,

-Duración,

-Tono (pitch),

-Canal correspondiente.

MidLib se encarga internamente de crear los eventos Note On y Note Off necesarios para representar cada nota según el tiempo y la duración proporcionada.

Una vez que todas las pistas ya hayan sido pobladas con eventos musicales e instrumentos, la libreia MidLib se encarga de toda la lógica de bajo nivel para poder convertir esta información en un archivo ".mid", esto incluye:

-Codificación del encabezado del archivo (tipo, número de pistas, resolución temporal),

-Serialización de cada pista con su secuencia de eventos ordenados por tiempo,

-Conversión a formato binario MIDI válido (según la especificación oficial),

-Escritura directa del archivo a disco, generando "salida.mid".

Gracias a esta abstracción, el programador no necesita manipular bits, bytes ni estructuras binarias del protocolo MIDI, ya que MidLib lo hace automáticamente con una interfaz sencilla y robusta.

Algorithm 5 Etapa 5 - Generación de Archivo MIDI

```
Crear objeto Sample con N pistas
for cada pista do
    Seleccionar pista activa
    Asignar canal e instrumento
    for cada nota do
        Insertar evento MIDI con tiempo, duración, pitch y canal
    end for
end for
Guardar el archivo como salida.mid
```

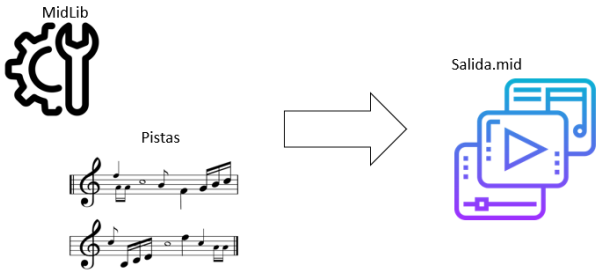


Fig. 5. Representation Etapa 4

El archivo generado "salida.mid" puede abrirse en cualquier software compatible con el estándar MIDI, el contenido de este archivo reflejara fielmente la partitura simbólica definida por el usuario con sus características principales como:

-Duraciones exactas de cada figura musical,

-Pausas correspondientes a los silencios,

-Cambios de octava aplicados correctamente,

-Repeticiones expandidas,

-Timbres instrumentales definidos por pista.

Todo esto garantiza una traducción precisa del texto de entrada a una obra musical digital, que respeta la intención del usuario y permite su reproducción y edición con herramientas modernas.

4. Comparación

Existen diversas herramientas que permiten generar música mediante código, cada una de estas con diferentes enfoques y objetivos distintos, entre estos programas destaca Sonic Pi, una plataforma educativa orientada al live coding.
Sonic Pi es usado para la creación e interpretación musical en tiempo real mediante una sintaxis basada en Ruby. Su objetivo es educativo y performativo, ofreciendo una experiencia interactiva donde el usuario puede modificar el código durante la ejecución y escuchar los resultados al instante.

En contraste el compilador musical propuesto sigue un modelo de compilación tradicional donde el archivo de entrada generar un archivo .mid, este enfoque permite una producción offline

de música, con una salida que puede ser cargada, editada o reproducida en cualquier estación de trabajo musical digital (DAW).

Otra diferencia clave es que Sonic Pi no genera archivos MIDI directamente, sino que produce sonido en tiempo real a través de un motor de síntesis (SuperCollider). Nuestro compilador, en cambio, trabaja a nivel simbólico con notación musical, generando eventos MIDI desde cero, lo que ofrece un control preciso sobre elementos como notas, duración, silencios, compases y octavas.

A continuación, se mostrarán las principales diferencias entre Sonic Pi y el compilador propuesto.

Característica	Compilador Propuesto	Sonic Pi
Enfoque	Compilar lenguaje musical personalizado a archivo .mid	Live coding (música en tiempo real)
Facilidad de uso	Requiere entender la sintaxis	Intuitivo para aprender de música programando
Uso de CPU (ejecución)	En ejecución muy bajo, solo requiere procesamiento y guardado de archivos	En ejecución alto en ejecución activa (síntesis + tiempo real)
Uso de CPU (compilación)	En compilación moderado, depende de la complejidad del usuario	No compila, ejecuta directamente en runtime
Ram	Baja (maneja estructuras de tokens y vectores simples)	Moderada a alta (carga motor de audio en memoria)
Salida	Archivo MIDI	Sonido directo

El enfoque de ambos programas es muy diferente, el compilador propuesto describe que debe sonar y que no, es muy útil para la generación estructurada, repetitiva o basada en patrones, mientras que el Sonic Pi es interactivo y performativo, diseñado para performances en vivo, enseñanza y exploración creativa.

El compilador propuesto a diferencia de Sonic Pi solo usa CPU para parsear el texto y generar notas, además de escribir el archivo .mid.

En cambio, Sonic Pi usa el motor de audio SuperCollider, que corre en segundo plano Al reproducir loops o efectos en vivo, puede usar entre 15%–50% del CPU fácilmente, dependiendo del número de procesos musicales activos.

5. Resultados

La experimentación con el compilador propuesto se realizo usando fragmentos de la canción I Was Made for Lovin' You, de la banda Kiss, el lenguaje diseñado permite representar notas, figuras rítmicas, compases y repeticiones de forma estructurada y comprensible, acercándose a una notación musical simbólica.

La siguiente figura muestra el pentagrama que se intenta replicar.

(re,n) (mi,n) (fa,n) (sol,c) (la,n) (la,c) (sol,n) (fa,n) (mi,n)
(re,n) (mi,n) (fa,n) (sol,c) (la,n) (la,c) (sol,b).

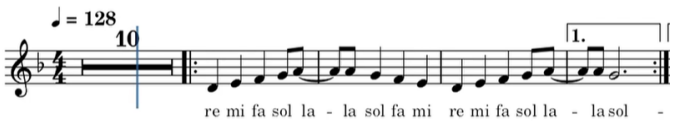


Fig.6 Representación de canción

En la figura 6 se representa un patrón melódico de la canción, en el pentagrama se define un patrón rítmico dentro de un compás 4/4, utilizando duraciones como negras y corcheas, además de aplicar repeticiones. Tras el análisis del código, el compilador realiza las tres fases clásicas del proceso de compilación (léxica, sintáctica y semántica), identificando tokens, validando estructuras gramaticales y generando eventos MIDI según lo descrito.

El archivo resultante de la compilación es un .mid estándar que puede ser reproducido y editado en cualquier estación de trabajo de audio digital (DAW). La validación semántica también está implementada para detectar errores como duraciones incompatibles con el compás, ausencia de delimitadores o invocaciones de patrones no definidos. En este caso, al tratarse de una entrada correcta, la compilación finaliza exitosamente generando una línea melódica coherente y fiel a la estructura rítmica esperada de la obra original.

6. Conclusiones

El compilador musical propuesto ofrece una alternativa estructurada y eficiente frente a diferentes herramientas interactivas, si bien ambas soluciones permiten generar música, el compilador propuesto sigue un pipeline clásico, generando archivos MIDI desde una representación textual precisa y controlada.

Gracias a este diseño, el compilador permite una producción musical offline, portable y compatible con entornos profesionales de edición musical. Además, su lenguaje específico, sencillo y centrado en la música tonal lo hace

accesible para usuarios con conocimientos básicos de programación, sin sacrificar flexibilidad ni expresividad en la composición.

El compilador propuesto se posiciona como una herramienta mas practica para compositores, estudiantes y desarrolladores que necesiten transformar texto estructurado en música digital reutilizable, distinguiéndose por su bajo consumo de recursos, facilidad de integración y compatibilidad con estándares de la industria como el formato .mid.

6. Referencias

[1] E. G. G. de Paz, P. M. Quintero Flores, y X. Quiñones Solís, "Lenguaje de programación para la composición automática de música," Programación Matemática y Software.

[2] D. P. Anderson y J. Bilmes, "Concurrent real-time music in C++," International Computer Science Institute, Berkeley, CA, USA.

[3] C. Ince, Programming for Music: Explorations in Abstraction, Master's thesis, School of Music, Media and Humanities, Univ. of Huddersfield, Huddersfield, UK, 2019.

[4] G. Loy y C. Abbott, "Programming languages for computer music synthesis, performance, and composition," Computer Audio Research Laboratory, Center for Music Experiment and Related Research, University of California, San Diego, CA, USA, y Lucasfilm Ltd., San Rafael, CA, USA.

[5] P. S. Langston, "Six techniques for algorithmic music composition," Bellcore, Morristown, NJ, USA.

[6] Pessoa, L. E. F., & Schiavoni, F. L. (2015, noviembre). Compilador musical: Uso da tecnologia de compiladores para validação de música simbólica. En 15º Simpósio Brasileiro de Computação Musical (Campinas, São Paulo).

[7] Galiano Ruiz, P. (2017, octubre). Sintetizador digital que almacena loop [Trabajo final de grado, Universitat Politècnica de Catalunya]. UPCommons.

[8] Rodríguez García, M. V. (2018). Modelos de enseñanza del lenguaje musical [Trabajo académico, Universidad Complutense de Madrid]. E-Prints UCM.

[9] Benítez, M. Á., Díaz Abrahan, V. M., & Justel, N. R. (2017). Beneficios del entrenamiento musical en el desarrollo infantil: Una revisión sistemática. Revista Internacional de Educación Musical, (5), 61–69. Sociedad Internacional para la Educación Musical.

[10] Cuervo, L., & Ordóñez, X. (2021). Beneficios de la estimulación musical en el desarrollo cognitivo de estudiantes de grado medio. Estudios Pedagógicos, 47(2), 339–353.

[11] Gutiérrez Aldana, A., & Tonche García, R. J. (2010). Diseño y desarrollo de un compilador visual para la enseñanza de la robótica básica [Tesis de maestría, Instituto Politécnico Nacional, Centro de Investigación en Computación]. Repositorio Digital IPN.

[12] Luzuriaga Aguilar, Á. (2021, 18 de octubre). Herramienta de conversión de notación de acordeón diatónico a partitura universal: Compilador y aplicación web [Trabajo académico, Universidad del País Vasco]. ADDI.

[13] Mathews, M. V. (1961). An acoustic compiler for music and psychological stimuli. Bell System Technical Journal, 40(3), 677–694.

[14] Jacobs, R., Feldmeier, M., & Paradiso, J. A. (2008). A mobile music environment using a PD compiler and wireless sensors [Informe técnico, Responsive Environments Group, MIT Media Lab]. MIT Media Lab.

[15] Davies, M., & Plumbley, M. D. (2005). Hang the DJ: Automatic sequencing and seamless mixing of dance music tracks. In 2005 IEEE Workshop on Applications of Signal Processing to Audio and Acoustics (pp. 303-306).