# Universidad Nacional de Quilmes

Escuela de Artes

LICENCIATURA EN MÚSICA Y TECNOLOGÍA

Director de Carrera: Esteban Calcagno

Programa de Investigación

CARTOGRAFÍAS ESPACIO-TEMPORALES Y ARTE SONORO

Director: Pablo Riera


Seminario de Investigación

## REPRESENTACIÓN TEXTUAL DE ESTRUCTURAS MUSICALES Y ENTORNO DE SECUENCIACIÓN


Presentado por: Lisandro Fernández



**Resumen**

Se propone un lenguaje formal basado en texto plano, descriptivo y serializado, capaz de representar informacion musical. Contextualiza, un conjunto de utilidades cuyo fin es producir secuencias musicales en el estándar MIDI.



Noviembre 2019

Buenos Aires, Argentina

# Contendios

# 1. Resumen

El presente trabajo de investigación propone un contexto de producción musical puramente textual mediante la definición de motivos musicales y la descripción de sus relaciones en combinación con un conjunto de herramientas informáticas.

En gran parte este proyecto se concentró en definir un marco de patrones y relaciones que permitan la representación sintáctica de información con significado musical, describiendo textualmente relaciones de referencia entre estructuras.

Por otro lado este estudio produjo un entorno de piezas de software para acompañar la propuesta. Se escribió un encadenado de procesos y manipulaciones que consume información contenida en archivos de texto serializado[1] y autodescriptivo subscrita a dicha definición y produce secuencias de mensajes con el fin de controlar dispositivos sonoros, tanto virtuales como físicos.

La primera parte de este escrito está dedicada a justificar el objeto de estudio, presentar los motivos de las interrogantes, plantear la necesidad de alternativas, también se discuten antecedentes en codificación textual de información musical.

En la segunda sección se describe el método de ejecución, detallando el procedimiento de desarrollo.

La parte central de este trabajo versa sobre el vocabulario y relaciones que conforman la gramática propuesta, se explica como dicha representación habilita que la semántica musical pueda ser materia prima de una serie de procesos y se despliega el resultado de algunos ejemplos a modo de demostración.

Para concluir se proyectan algunas aplicaciones posibles en diferentes escenarios (trabajo colaborativo en simultaneo y a distancia, programación en vivo) y varias disciplinas (inteligencia artificial, archivología).

Completa el aspecto técnico de este trabajo la inclusión del código de los módulos desarrollados para la implementación.

# 2. Introducción

En esta sección inaugural se enmarca la investigación, argumentando la constricción principal, la adopción de un sistema de escritura como contenedor de instrucciones y medio de interacción.

Seguido se repasan las necesidades que denotan la pertinencia de este estudio, aludiendo a requerimientos externos a satisfacer.

Para concluir esta introducción se tratan trabajos semejantes de cierta relevancia a este proyecto.

[1] Coombs, Renear y DeRose (1987)

## 2.1. Justificación

Se repasan las ventajas principales del registro de información con enunciados textuales y del empleo del lenguaje como medio de entrada de instrucciones en escenarios generales.

### 2.1.1. Texto Llano

> "…our base material isn't wood or iron, it's knowledge. […]. And we believe that the best format for storing knowledge persistently is plain text. With plain text, we give ourselves the ability to manipulate knowledge, both manually and programmatically, using virtually every tool at our disposal." (Hunt y Thomas 1999)

Se listan las virtudes del texto plano y legible en contraste a la codificación binaria de datos[2] o cualquier otro tipo de operación que opaque la relación con lo representado

**Mínimo Común Denominador.** Potencialmente cualquier herramienta de computo puede operar información almacenada en texto plano. Es soportado en múltiples plataformas, cada sistema operativo cuenta con al menos un editor de texto todos compatibles hasta la codificación.

**Fácil de manipular.** Procesar cadenas de caracteres es de los trabajos mas rudimentarios que pueden ser realizados por un sistema informático.

**Fácil de mantener.** El texto plano no presenta ninguna dificultad o impedimento ante la necesidad de actualizar información o de realizar cualquier tipo de cambio o ajuste.

**Fácil de comprobar.** Es sencillo agregar, actualizar o modificar datos de testeo sin la necesidad de emplear o desarrollar herramientas especiales para ello.

**Liviano.** Determinante cuando los recursos de sistema son limitados como por ejemplo almacenamiento escaso, velocidad de computo restringida o conexiones lentas.

**Seguro contra la obsolescencia, o compatible con el avance.** Los archivos de datos en formatos legibles y autodescriptivos perduran por sobre otros formatos aun cuando caduquen las aplicaciones que las hayan creado.[3]

### 2.1.2. Intérprete de Comandos

Se argumenta la conveniencia de prescindir de representaciones gráficas como canal de interacción con herramientas informáticas.

[2] Hunt y Thomas (1999) Capítulo 3: Basic Tools (pp. 72-99).
[3] Leek (2017)

**Estado operativo de un ordenador inicial.** Eventualmente todos los sistemas operativos permiten ser utilizados a través de este acceso previo al gerente de escritorio.

**Menor utilización de recursos.** No depender de un agente de ventanas interviniendo entre el usuario y el sistema libra una cantidad considerable de recursos.

**Una única interfaz para multiples aplicaciones.** La estructura de instrucciones esperada en esta interfaz, *aplicación - argumento - recurso* (su analogía *verbo - adverbio - sujeto*), persiste para cualquier pieza de software. Dicha recurrencia elimina el ejercicio que significa un operar distinto para cada aplicación, fomentando un accionar similar en contextos y circunstancias diferentes.

**Tradición.** Perdura por décadas como estándar durante la historia de la informática remitiendo a los orígenes de los ordenadores basados en teletipo.

**Resultados reproducibles.** Si bien la operación de sistemas sin más que la entrada de caracteres requiere conocimiento y entrenamiento específico, no considerar la capa que representa la posición del puntero como parámetros de instrucciones, permite que sean recopiladas en secuencias de acciones precisas y reutilizar estos guiones en diferentes escenarios, con diferentes agentes.

**Encadenado y Automatización.** La posibilidad de componer rutinas complejas de manipulación concatenando resultados con procesos.[4]

**Gestión remota.** Mas allá del protocolo en el que se base la negociación local/remoto la interfaz de linea de comandos es la herramienta de facto para administrar un sistema a distancia.

**Productividad.** Valerse de herramientas pulidas como editores de texto avanzados que gracias al uso de atajos (acciones complejas asignadas a combinaciones de teclas) evitan la alternancia entre mouse y teclado, lo cual promueve un flujo de trabajo ágil.[5]

Siendo estas razones de carácter general, las mismas aplican al propósito particular que implica este estudio.

### 2.1.3. Interfaz Digital para Instrumentos Musicales (MIDI)

De carácter especifico a la producción musical, en relación directa a este proyecto es menester acreditar la adopción de un protocolo en particular para codificar la capa que describe y gestiona la performance entre dispositivos.[6]

[4] Raymond (1999) Capítulo 1: Context, Apartado 1: Philosophy, Sub-apartado: Basics of the Unix Philosophy (pp. 34-50)

[5] Moolenaar (2000)

[6] Haus y Ludovico (2007)

El animo por que las secuencias de control a producir satisfagan las condiciones requeridas para alcanzar compatibilidad con el protocolo MIDI estándar[7], está fundamentado por sus virtudes de ser y proyectarse ampliamente adoptado, soportado en la mayoría de los entornos y apoyado por la industria.[8]

Si bien es ágil y se planea compatible a futuro, cualidades que comparte con el formato de texto llano, es ineludible la desventaja que significa el empleo de cualquier sistema de codificación[9], intrínseco a la dificultad que impone para interpretar a simple vista la información cifrada, ofuscación que resulta en la dependencia de herramientas especificas para cualquier manipulación.

## 2.2. Motivación

Este proyecto plantea la necesidad de establecer un contexto y proveer recursos para un procedimiento sencillo pero a la vez ágil y flexible de elaboración discursos musicales acercando la planificación de obra a la secuenciación MIDI.

Ademas pretende exponer las ventajas de la Interfaz de Linea de Comandos para operar herramientas informáticas a la comunidad de artistas, teóricos e investigadores.

Promover la adopción de prácticas consolidadas y formatos abiertos para representar, manipular y almacenar información digital.

Fomentar el trabajo colaborativo generando vínculos con y entre usuarios.[10][11]

## 2.3. Antecedentes

A continuación se describen algunos desarrollos que vinculan representación y manipulación de información musical: MuseData, Humdrum, MusicXML y MML; como ejemplo de un marco de programación basada en una sintaxis declarativa se consideró Flocking.

### 2.3.1. MuseData

La base de datos MuseData[12] es el sistema de codificación principal del Centro de Investigación Asistida por Computador en Humanidades (CCARH) de la Universidad de Stanford. La base de datos fue creada por Walter Hewlett.

Los archivos MuseData tienen el potencial de existir en múltiples formatos comunes de información. La mayoría de las codificaciones derivadas definen sólo algunas de las las características incluidas en el master MuseData de codificaciones. El archivo MuseData está diseñado para soportar aplicaciones de sonido,

[7] (MMA) (1996)
[8] Penfold (1992)
[9] Cifrado condicionante para el trasporte.
[10] Raymond (1997) Capítulo 11: The Social Context of Open-Source Software (p. 11)
[11] Yzaguirre (2016)
[12] Selfridge-Field (1997) Capitulo 27: MuseData: Multipurpose Representation

gráficos y análisis. Los formatos derivados de las codificaciones musicales de MuseData que se distribuyen son: MIDI1, MIDI+ y Humdrum.

### Organización de archivos MuseData

Los archivos MuseData están basados en ASCII y se pueden ver en cualquier editor de texto. Dentro del formato MuseData el número de archivos por movimiento y por trabajo puede variar de una edición a otra. Estos ficheros están organizados en base a las partes. Un movimiento de una composición es típicamente encontrado dividido en varios archivos agrupados en un directorio para ese movimiento.

### La representación MuseData de información musical

El propósito de la sintaxis MuseData es representar el contenido lógico de una pieza musical de un modo neutral. El código se utilizó en la construcción de bases de datos de texto completo de música de varios compositores, J.S. Bach, Beethoven, Corelli, Handel, Haydn, Mozart, Telemann y Vivaldi. Se pretende que estas bases de datos de texto completo se utilicen para la impresión de partituras, análisis musical y producción de archivos de sonido digitales.

Aunque el código MuseData está destinado a ser genérico, se han desarrollado piezas de software de diversos tipos con el fin de probar su eficacia. Las aplicaciones MuseData pueden imprimir resultados y partes para ser utilizadas por editores profesionales de música, así como también compilar archivos MIDI (que se pueden utilizar con secuenciadores estándar) y facilitar las búsquedas rápidas de los datos de patrones rítmicos, melódicos y armónicos específicos.

La sintaxis MuseData está diseñada para representar tanto información de notación como de sonido, pero en ambos casos no se pretende que la representación esté completa. Eso prevé que los registros MuseData servirían como archivos de origen para generar tanto documentos gráficos (específicamente de página) y archivos de performance MIDI, que podrían editarse como el usuario lo crea conveniente. Las razones de esta postura son dos:

- Cuando se codifica una obra musical, no es la partitura sino el contenido lógico de la partitura lo que codifica. Codificar la puntuación significaría codificar la posición exacta de cada nota en la página; pero es probable que tal codificación realmente contenga más información que la que el compositor pretende transmitir.
- No se puede anticipar todos los usos que podrían darse a estos datos, pero si asegurar que cada usuario tendrá sus propias necesidades especiales y preferencias. Por lo tanto, no tiene sentido tratar de codificar información acerca de cómo debe verse una representación gráfica de los datos o cómo estos datos deben sonar.

Por otro lado, a veces puede ser útil hacer sugerencias sobre cómo los gráficos y el sonido deben ser realizados. Lo importante es identificar las sugerencias como

un tipo de dato independiente, que puede ser fácilmente ignorado por software de aplicación o despojado enteramente de los datos. MuseData usa estas sugerencias de impresión y sonido en el proceso de generación de documentos de partitura y archivos MIDI.

### 2.3.2. Humdrum

David Huron creó Humdrum[13] en los años 80, y se ha utilizado constantemente por décadas. Humdrum es un conjunto de herramientas de línea de comandos que facilita el análisis, así como una sintaxis generalizada para representar secuencias de datos. Debido a que el conjunto de herramientas es de lenguaje agnóstico, se han empleado herramientas de Humdrum en secuencias de comandos más grandes que utilizan PERL, Ruby, Python, Bash, LISP y C++.

#### Representación

En primer lugar, Humdrum define una sintaxis para representar información discreta como una serie de registros en un archivo de computadora.

Esta definición permite que se codifiquen muchos tipos de información. El esquema esencial utilizado en la base de datos CCARH para la altura y la duración musical es sólo uno de un conjunto abierto. Algunos otros esquemas pueden ser aumentados por gramáticas definidas por el usuario para tareas de investigación.

#### Manipulación

Segundo, está el conjunto de comandos, el Humdrum Toolkit, diseñado para manipular archivos que se ajusten a la sintaxis Humdrum en el campo de la investigación asistida por ordenador en la música.

El énfasis está en **asistido**:

- Humdrum no posee facultades analíticas de nivel superior per se.
- Más bien, *su poder deriva de la flexibilidad de su kit de elementos, utilizados en combinacióin* para aprovechar plenamente el potencial del sistema.

#### De la experiencia a la apreciación

Apreciación de todo el potencial de Humdrum es definitivamente a partir de la experiencia. En palabras de David Huron:

> Cualquier conjunto de herramientas requiere el desarrollo de una experiencia concomitante, y Humdrum Toolkit no es una excepción. Espero que la inversión del tiempo requerido para aprender a usar Humdrum sea más que compensado por ganancias académicas posteriores.

[13] Wild (1996)

Los usuarios de Humdrum hasta ahora han tendido a trabajar en la percepción de la música o etnomusicología, mientras que los teóricos y los musicólogos histioriadores han sido mas lentos para reconocer el potencial del sistema.

#### CLI vs GUI

Humdrum u otros sistemas como él ofrecen los recursos para marcar un paradigma en la investigación musical.

El tedio de recopilar pruebas sólidas que apoyen las propias teorías pueden ser aliviadas por la automatización y cuanto mayor sea la cantidad de música examinada mayor será el rigor de la prueba de las hipótesis.

Sin embargo, la desafortunada posibilidad es que muchos de los musicólogos y teóricos que se beneficiarían de una pequeña intuición asistida por la máquina, sean repelidos por la interfaz totalmente basada en texto de Humdrum.

Aunque en el análisis final los comandos estilo UNIX son seguramente más flexibles y eficientes que una interfaz gráfica "amigable", pueden intimidar a principiantes, muchos de los cuales pueden resultar disuadidos de emplear herramientas de utilidad considerable.

Independientemente que teóricos de la música decidan o no aumentar su intuición musical con valiosas pruebas empíricas, los resultados basados en las cantidades máximas de datos pertinentes será un factor en la evolución de nuestra disciplina.

### 2.3.3. MusicXML

MusicXML[14] fue diseñado desde cero para compartir archivos de música entre aplicaciones y archivar registros de música para uso en el futuro. Se puede contar con archivos de MusicXML que son legibles y utilizables por una amplia gama de notaciones musicales, ahora y en el futuro. MusicXML complementa formatos de archivo utilizados por Finale y otros programas.

MusicXML pretende ser un estándar para compartir partituras interactivas, dado que facilita crear música en un programa y exportar sus resultados a otros programas. Al momento más de 220 aplicaciones incluyen compatibilidad con MusicXML.

### 2.3.4. Music Markup Language

El Lenguaje de Marcado de Música (MML)[15] es un intento de marcar objetos y eventos de música con un lenguaje basado en XML. La marcación de estos objetos debería permitir gestionar música en documentos para diversos fines, desde la teoría musical y la notación hasta el rendimiento práctico. Este proyecto no está completo y está en progreso. El primer borrador de una posible DTD

[14] Good (2001)
[15] Steyn (2001)

está disponible y se ofrecen algunos ejemplos de piezas de música marcadas con MML.

El enfoque es modular, varios módulos aún están incompletos y necesitan más investigación y atención. Una pieza musical serializada usando MML puede ser entregada en al menos los siguientes formatos:

- Texto: representación de notas como, por ejemplo, piano-roll (como el que se encuentra en el software del secuenciador de computadora).
- Common Western Notation: Notación musical occidental en pantalla o en papel
- MIDI-device: MML hace posible "secuenciar" una pieza de música sin tener que usar software especial. Así que cualquier persona con un editor de texto debe ser capaz de secuenciar la música de esta manera.

### 2.3.5. Flocking

Flocking[16] es un framework, escrito en JavaScript, para la composición de música por computadora que aprovecha las tecnologías e ideas existentes para crear un sistema robusto, flexible y expresivo. Flocking combina el patrón generador de unidades de muchos idiomas de música de computadora con tecnologías Web Audio para permitir a los usuarios interactuar con sitios Web entre otras potenciales tecnologías, usando un estilo declarativo de programación.

El objetivo de Flocking es permitir el crecimiento de un ecosistema de herramientas que puedan analizar y entender fácilmente la lógica y la semántica de los instrumentos digitales representando de forma declarativa los pilares básicos de síntesis de audio. Esto es particularmente útil para soportar la composición generativa donde los programas producen nuevos instrumentos de forma algorítmica, herramientas gráficas para que programadores y no programadores colaboren, y nuevos modos de programación social que permiten a los músicos adaptar, ampliar y volver a trabajar fácilmente en instrumentos existentes.

#### Programación declarativa

Arriba, se describió Flocking como un marco **declarativo**. Esta característica es esencial para comprender su diseño. La programación declarativa se puede entender en el contexto de Flocking por dos aspectos esenciales:

1. Enfatiza una visión semántica de alto nivel de la lógica y estructura de un programa
2. Representa los programas como estructuras de datos que pueden ser entendido por otros programas.

[16] Clark y Tindale (2014)

El énfasis aquí es sobre los aspectos lógicos o semánticos de la computación, en vez de en la secuenciación de bajo nivel y el flujo de control. Tradicionalmente los estilos de programación imperativos suelen estar destinados solo para el compilador. Aunque el código es a menudo compartido entre varios desarrolladores, no suele ser comprendidos o manipulados por programas distintos a los compiladores.

Por el contrario, la programación declarativa implica la capacidad de escribir programas que están representados en un formato que pueden ser procesados por otros programas como datos ordinarios. La familia de lenguajes Lisp es un ejemplo bien conocido de este enfoque. Paul Graham describe la naturaleza declarativa de Lisp, expresando que "no tiene sintaxis. Escribes programas en árboles de análisis… [que] son totalmente accesibles a tus programas. Puedes escribir programas que los manipulen… programas que escriben programas".[17] Aunque Flocking está escrito en JavaScript, comparte con Lisp el enfoque para expresar programas dentro de estructuras de datos que estén disponibles para su manipulación por otros programas.

Si bien la recopilación expuesta no agota la lista de referentes pertinentes y otros que cobraran relevancia, provee un criterio inicial para proceder.

# 3. Metodología

En este capitulo se introduce el procedimiento de ejecución en el que se pueden distinguir dos etapas, una preparatoria dedicada a investigación, experimentación y pruebas, seguida consecuentemente por la fase de producción.

Se aprovecha para reseñar herramientas preexistentes elegidas, se mencionan aquellas que fueron consideradas pero descartadas luego de algunos ensayos y otras periféricas vinculadas a la tarea accesoria.

## 3.1. Preliminares

Se explican experiencias tempranas necesarias para evidenciar y comprobar que la hipótesis formulada fuese al menos abarcable y fundamentar los pasos siguientes.

A partir de las inquietudes presentadas, se propuso como objetivo inicial establecer una lista de parámetros que asocien valores a propiedades musicales elementales (altura, duración, intensidad, etc) necesarias para definir el conjunto de articulaciones constituyentes de un discurso musical, en determinado sentido rudimental, austero y moderado.

Acorde a esto se hilvanó una rutina de procesos, compuesta por un analizador sintáctico y un codificador digital entre otras herramientas, que a partir de valores emita un flujo de mensajes.

[17] Graham (2001)

### 3.1.1. Boceto de Gramática

El método para discretizar información, jerarquizar y distinguir propiedades de valores, se basa en el formato YAML.[18] Luego de considerar este estándar y enfrentarlo con alternativas, se concluye que cumple con las condiciones y que es idóneo para la actividad.

Implementaciones del mismo en la mayoría de los entornos vigentes[19], aseguran la independencia de la información serializada en este sistema. Se le adjudica alta legibilidad[20]. Goza de cierta madurez, por lo que fue sujeto de ajustes y mejoras[21].

### 3.1.2. Prototipo

Se esbozó un guión de instrucciones acotado a componer cadenas de eventos a partir de la interpretación, análisis sintáctico, proyección (mapeo) y asignación de valores.

Este prototipo, que confirmó la viabilidad de la aplicación pretendida, fue desarrollado en Perl,[22] lenguaje que luego de algunas consideraciones se desestimó por Python[23] debido principalmente a mayor adopción en la producción académica.

## 3.2. Desarrollo

En las actividades posteriores a las comprobaciones, aunque influenciados entre si, se pueden distinguir dos agrupamientos:

- Establecer relaciones de sucesión y jerarquía, que gestionen herencia de propiedades entre segmentos musicales subordinados o consecutivos y extender el léxico admitido con el propósito de cubrir una cantidad mayor de propiedades musicales.
- Escalar el prototipo a una herramienta que capaz de consumir recursos informáticos, interpretar series de registros, manipular valores, derivarlos en articulaciones, empaquetar y registrar secuencias, entre otras propiedades.

### 3.2.1. Lenguaje

Al establecer este lenguaje formal, el primer esfuerzo se concentró en definir la organización de las propiedades de cada parte musical, conseguir una estructura lógica que ordene un discurso multi-parte.

[18] Ben-Kiki, Evans y Ingerson (2005)
[19] (n.d.) (2019b)
[20] Huntley (2019)
[21] Ben-Kiki Oren y Ingy (2009)
[22] Wall (1999)
[23] Rossum (2018)

La discriminación de los datos comienza a nivel de archivo, cada fichero contiene los datos relativos a estratos individuales en la pieza. Obteniendo así recursos que canalizan la información de cada parte junto con determinadas propiedades globales (tempo, armadura de clave, metro, letras, etc), que si bien pueden alojarse en una definición de canal, son meta eventos[24] que afectaran al total de la pieza[25].

Dicho esto se continua con la organización interna de los documentos y algunas consideraciones acerca del léxico acuñado.

### Formato YAML

Las definiciones de pista son regidas por YAML. Si bien el vocabulario aceptado es propio de este proyecto, todas las interpretaciones son gestionadas por dicho lenguaje. Se reseñan los principales indicadores reservados y estructuras básicas.

En el estilo de bloques de YAML, similar al de Python, la estructura esta determinada por la indentación. En términos generales indentación se define como los espacios en blanco al comienzo de la linea. Por fuera de la indentación y del contexto escalar, YAML destina los espacios en blanco para separar entre símbolos.

Los indicadores reservados pertinentes a señalar son: los dos puntos ":" denotan la proyección de un valor, el guión "-" indica un bloque de secuencia, *ancla* el nodo para referencia futura el símbolo "&" ampersand, habilitando así inclusiones adicionales del mismo cuyos *alias*, apariciones subsecuentes, son invocados con el simbolo "*" asterisco.

Quizás esta presentación austera suscite una intimidación aparente, como se aprecia en los ejemplos desplegados en el capitulo siguiente, con algunas reglas sencillas este lenguaje de marcado consigue plena legibilidad, sin dejar de ser flexible ni expresivo. Para mas información acerca de otras estructuras y el tratamiento especial de caracteres reservados, referirse a la especificación del formato[26].

### Vocabulario

Con intención de favorecer a la comunidad hispanoparlante el léxico que integra este lenguaje específico de dominio[27] esta compuesto, salvo contadas excepciones, por vocablos del diccionario español. De todos modos, son sencillas las modificaciones requeridas para habilitar la comprensión de términos equivalentes (en diferentes idiomas).

[24] Selfridge-Field (1997) Capitulo 3: MIDI Extensions for Musical Notation (1): NoTAMIDI Meta-Events

[25] La limitación en cantidad de canales y el carácter global de algunas propiedades son algunas de las imposiciones del el estándar MIDI.

[26] Ben-Kiki, Evans y Ingerson (2005) Apartado 5.3: Indicator Characters y Capitulo 6: Basic Structures.

[27] (n.d.) (2019a)

Para negociar con la noción inabarcable que significa dar soporte a cada aspecto musical esperado, siendo imposible anticipar todos las aplicaciones estipuladas en determinado sentido arbitrarias y/o circunstanciales, se propone un sistema de complementos de usuarios que habilita la salida y entrada de valores, para su manipulación externa a la rutina provista. Si bien en el uso este sistema se mostró prometedor, su naturaleza no excede el carácter experimental y es menester promover mejoras y consideraciones adicionales.

Los componentes del léxico y el sistema de complementos son detallados en el primer apartado del capitulo siguiente.

### 3.2.2. Entorno

Tanto las abstracciones desarrolladas, así como también la rutina de instrucciones principales, están destinadas al intérprete *Python 3*[28]. Ademas de incorporar al entorno varios módulos de la "Librería Estándar"[29] esta pieza de software está apoyada en otros dos complementos, el marco de trabajo "PyYAML"[30] para asistir con el análisis sintáctico, en combinación con la librería "MIDIutil"[31] encargada de la codificación.

En el mismo ánimo con el que se compuso el vocabulario, el guion de acciones hace uso intensivo del idioma español. Esta decision es en cierto aspecto cuestionable siendo mayoritarias las sentencias predefinidas en ingles impuestas por el entorno.

#### Analizador Sintáctico

El primer proceso en la rutina es el de consumir información subscrita, interpretarla y habilitarla para su manipulación posterior. Se confía esta tarea al analizador sintáctico *PyYAML*.

En la presentación oficial del entorno dice:

- Ser completamente capaz de analizar YAML en su version 1.1, comprendiendo todos los ejemplos de dicha especificación.
- Implementar un algoritmo referente gracias a su sencillez.
- Soportar la codificación de caracteres Unicode en la entrada y la salida.
- Analizar y emitir eventos de bajo nivel, con la posibilidad alternativa de emplear la librería de sistema LibYAML.
- Poseer una interfaz de programación de alto nivel sencilla para objetos nativos Python. Con soporte para todos los tipos de datos de la especificación.

[28] Rossum (2018)
[29] (n.d.) (2018b)
[30] (n.d.) (2018a)
[31] Wirts (2016)

### Codificación de Salida

La cadena de procesos finaliza cuando la lista articulaciones resultante, hasta esta instancia abstracciones en memoria, es secuenciada en eventos, codificada y registrada en ficheros.

MIDIUtil es una biblioteca que posibilita generar piezas multi-parte en formato MIDI 1 y 2 desde rutinas de Python. Posee abstracciones que permite crear y escribir estos archivos con mínimo esfuerzo.

El autor escusa implementar selectivamente algunos de los aspectos más útiles y comunes de la especificación MIDI, argumentando tratarse de un gran documento en expansión a lo largo de décadas. A pesar de ser en parte incompleta, las propiedades cubiertas fueron suficientes para este proyecto y sirvió como marco el objetivo de dar soporte a cada aspecto comprendido por la librería[32].

### Otras herramientas

Para concluir el relato de método se mencionan dos herramientas accesorias de las cuales se hizo uso intensivo, tanto en el desarrollo de la investigación, como así también en la producción de este documento.

Siendo este proyecto texto-centrista, el ecosistema está incompleto sin un editor de texto apropiado[33]. Para conseguir fluidez y consistencia la herramienta empleada para esta actividad tiene que poder operar según el contexto, manipular bloques, disponer de macros sencillos y configurables. Para estos asuntos se confió en Vim[34].

El progreso y el respaldo en linea, fue agilizado por el sistema de control de versiones GIT.[35] Es esta la herramienta que permite conseguir el repositorio contenedor del desarrollo junto con instrucciones de instalación e indicaciones de uso[36].

Pese a que se comprenden estos temas en el dominio de usuario, se reconoce la ventaja y se sugiere el empleo de este tipo de herramientas.

## 4. Resultados

Se dedicada esta sección a detallar en profundidad el sistema propuesto. Comenzando por los constituyentes y preceptos, seguido los procesos de los cuales estos son objeto.

Concluyendo se exponen, definiciones de pista en sintaxis YAML como entrada y la representación gráfica de la salida en formato de partitura para dos ejemplos:

[32] Wirts (2016)
[33] Moolenaar (2000)
[34] Oualline (2001)
[35] Torvalds y Hamano (2010)
[36] Enlace al repositorio en linea: https://gitlab.com/lf3/yml2mid

una melodía sencilla y una extracto musical con multiples partes instrumentales.

## 4.1. Gramática

Sobre la estructura sentada por el formato YAML opera otro juego de reglas propio a este desarrollo que gobierna la combinatoria entre constituyentes. Luego de exponer estos principios se presenta el vocabulario concebido, que junto con la sintaxis completa esta gramática.

### 4.1.1. Sintaxis

El discurso musical de cada parte se organiza en dos niveles. Se distinguen las propiedades globales que afectan a la totalidad de la pista de las que en un siguiente dominio, definen cualidades particulares a cada unidad musical[37].

De la lista dispuesta en el próximo apartado, en cuestiones constitutivas se destaca el término `forma`[38]. Este indica la organización de unidades y recibe el mismo tratamiento a nivel macro que a nivel micro, en ambos casos representa una lista ordenada de referencias declaradas y disponibles en la paleta de `unidades`.

Si el elemento carece de este atributo ninguna otra unidad es invocada, por lo tanto se ejecuta el segmento.

Existe un secuencia de secciones musicales de primer grado que es relativa a la pista, de ahí en adelante las unidades se refieren entre ellas hasta alcanzar segmentos de ultimo grado, resultando una organización de árbol[39].

Previo a la descripción del vocabulario aceptado, relacionada a esta organización es la gestión de herencia entre unidades, la sucesión de propiedades. Unidades invocadas heredan las cualidades del referente, este sobrescribe propiedades en los referidos.

### 4.1.2. Léxico

A modo de referencia, se describe el léxico acuñado aprovechando la distinción expuesta anteriormente entre propiedades generales a la pista y particulares a las unidades.

Para detallar cada término evitando redundancias, se organiza la información repitiendo la misma estructura para cada uno de ellos. Se presentan en linea: el término que identifica la propiedad, el tipo de dato que se espera, el valor asignado por defecto, luego una breve descripción y por ultimo un ejemplo.

[37] Grela (1992) Se adopta la terminología *unidad* para referir elementos musicales y *grado* para denotar el alcance de dicho agrupamiento.

[38] Se utiliza fuente tipográfica `monoespaciada` para resaltar el vocabulario propio de este proyecto.

[39] Pope (1986) Designing Musical Notations, Sequences And Trees.

### Propiedades de Pista

Los parámetros generales de cada pista son cuatro: El `nombre` de la pista define el rotulo soportado por el estándar MIDI que identifica dicha parte en la pieza, la paleta de `unidades` disponibles, la `forma` indica la secuencia de unidades de primer grado y los `complementos` de usuario.

`nombre` (cadena de caracteres) default: `<empty>`

Titulo de la pista.

```
nombre: 'Feliz Cumpleaños'
```

`unidades` (diccionario) default: `[ None ]`

Paleta de unidades disponibles (con sus propiedades respectivas) para ser invocadas en `forma`.

```
unidades:
  base: &base
    transportar: 72
    registracion: [ 0, 2, 4, 5, 7, 9, 11, 12 ]
    metro: 3/4
  a: &a
    <<: *base
    alturas: [ 5, 5, 6, 5 ]
    duraciones: [ .75, .25, 1, 1 ]
  b: &b
    <<: *base
    alturas: [ 8, 7 ]
    duraciones: [ 1, 2 ]
  motivoA:
    forma: [ 'a', 'b' ]
  motivoB:
    forma: [ 'b', 'b' ]
```

`forma` (lista de cadenas de caracteres) default: `[ None ]`

Macro estructura de la pista. Lista de unidades a ser secuenciadas, cada elemento corresponde a un miembro de la paleta de `unidades`.

```
forma: [ 'motivoA', 'motivoB' ]
```

`complementos` (cadena de caracteres) default: `None`

Ubicación de módulo con métodos de usuario.

```
complementos: 'enchufes.py'
```

### Propiedades de Unidad

En el diccionario de unidades de la pista cada entrada representa una unidad disponible, que a su vez aloja sus cualidades. Esta es la lista de términos aceptados como propiedades para cada unidad.

`forma` (lista de cadenas de caracteres) default: `[ None ]`

Estructura de la unidad. Lista de unidades referidas a ser secuenciadas.
Cada elemento corresponde a un miembro de la paleta de unidades.

```
forma: ['A', 'B']
```

`alteraciones` (número entero) default: 0

Cantidad de alteraciones en la armadura de clave.
Números positivos representan sostenidos y números negativos indican la cantidad bemoles.

```
alteraciones: -2 # Bb
```

`modo` (número entero) default: 0

Modo de la escala.
El número 0 indica que se trata de una escala mayor, mientras que el 1 representa tonalidad menor.

```
modo: 1 # menor
```

`registracion` (lista de enteros) default: `[ 1 ]`

Conjunto de intervalos a ser indexados por el puntero de `alturas`.

```
registracion: [
  -12,-10, -9, -7, -5, -3, -2,
    0,  2,  3,  5,  7,  9, 10,
   12, 14, 15, 17, 19, 21, 22,
   24
]
```

`transportar` (número entero) default: 0

Ajuste de alturas en semitonos

```
transportar: 60 # C
```

`transponer` (número entero) default: 0

Transponer puntero de intervalo.
Ajuste de alturas, pero dentro de la `registracion` fija.

```yaml
transponer: 3
```

`metro` (cadena de caracteres) default: `4/4`

Clave de compás.

```yaml
metro: 4/4
```

`reiterar` (número entero) default: `1`

Repeticiones, cantidad de veces q se toca esta unidad.
Esta propiedad no es transferible, no se sucede (de lo contrario se reiteraran los referidos).

```yaml
reiterar: 3
```

`canal` (número entero) default: `1`

Número de Canal.

```yaml
canal: 3
```

`afinacionNota` (diccionario) default: `None`

Afinación de nota.

```yaml
afinacionNota:
  afinaciones: [ [ 69, 50 ], [ 79, 60 ] ]
  canalSysEx: 127
  tiempoReal: true
  programa: 0
```

`afinacionBanco` (diccionario) default: `None`

Afinación banco.

```yaml
afinacionBanco:
  banco: 0
  ordenar: false
```

`afinacionPrograma` (diccionario) default: `None`

Afinacion Programa.

```yaml
afinacionPrograma:
  programa: 0
  ordenar: false
```

`RPN` (diccionario) default: `None`

Realizar llamada a parámetro numerado registrado.

```
RPN:
  control_msb: 0
  control_lsb: 32
  data_msb: 2
  data_lsb: 9
  ordenar: True
```

`NRPN` (diccionario) default: `None`

Realizar llamada a parámetro numerado no registrado.

```
NRPN:
  control_msb: 0
  control_lsb: 32
  data_msb: 2
  data_lsb: 9
  ordenar: True
```

`sysEx` (diccionario) default: `None`

Agregar un evento de sistema exclusivo.

```
sysEx:
  fabricante: 0
  playload:  !!binary ''
```

`uniSysEx` (diccionario) default: `None`

Agregar un evento de sistema exclusivo universal.

```
uniSysEx:
  codigo: 0
  subCodigo: 0
  playload:  !!binary ''
  canal: 14
  tiempoReal: False
```

`metodo-usuario` (diccionario) default: `None`

Invocar métodos declarados como `complementos` de usuario mediante su respectivo nombre, subordinando a este las propiedades de unidad que se quieran manipular y pasar como valores los argumentos que espera esta rutina.

```
unidades:
  a:
```

```
alturas: [ 5, 5, 6, 5 ]
fluctuar: # metodo-usuario
  dinamicas: .5 # propiedad:argumentos
  duraciones: .3 # propiedad:argumentos
desplazar: # metodo-usuario
  duraciones:  .25 # propiedad:argumentos
```

### Articulaciones

Si bien estructuralmente no se distingue otra jerarquía, las siguientes propiedades actúan a nivel de articulación, se subscriben a unidades pero en vez de modificar al segmento como conjunto, resultan en un valor por cada articulación.

Comparten la cualidad de esperar listas de valores y el proceso combinatorio al cual son sujetas es similar al empleado en la técnica compositiva del motete isorrítmico[40], difiriendo en que el procedimiento no se limita a duraciones y alturas, abarca otras propiedades.

La cantidad de articulaciones producidas es equivalente al número de miembros en la serie mas extensa, se reiteran secuencialemente patrones mas cortos alineandose, hasta completar el total de articulaciones.

Es pertinente señalar la combinatoria que resulta indexando los valores de la serie de `alturas` como punteros en el conjunto intervalos de `registración` fija.

## 4.2. Implementación

En este apartado se expone la estructura de la aplicación, se dispone un esquema, de carácter introductorio, del flujo de procesos y seguido se detallan las funciones principales y conexiones de cada componente.

Antes de la descripción de cada capa de abstracción, con intención de facilitar la comprensión, se presenta un esquema de la cadena de procesos de la rutina superior y relaciones internas.

```
YAML     \ parser       | Secuencia      | codificador / MIDI
archivos > definiciones > Pistas         > tracks      /
-------- \ ------------ | -------------- | ----------- /
         \     Unidades = Secciones      |             /
         \     Unidades = Segmentos      = llamadas    /
         \              | Articulaciones = llamadas    /
........ \ ............ | ............... | ........... /
```

[40] Variego (2018)

La rutina principal comienza leyendo, desde argumentos posicionales, definiciones de pista. Después de analizadas sintacticamente se entregan como diccionarios reunidos en un lista al modulo *Secuencia*.[41]

El producto de dicha manipulación es una lista ordenada de llamadas a diversas funciones que operan el codificador, el cual concluye el programa emitiendo una secuencia de mensajes MIDI.

A continuación se expone el trazado de mecanismos internos, desde el nivel de abstracción superior *Secuencia*, alcanzando la capa *Articulación* comprendida como la mas profunda.

### 4.2.1. Secuencia

Después que cada nodo en los datos recibidos es consolidado como objeto de clase *Pista*, esta instancia finaliza recorriendo articulaciones y cambios de parámetros entre los elementos, reuniendo todos los pronunciamientos en un único flujo de instrucciones[42].

#### Pista

Este módulo es responsable por el devenir del las partes musicales. Desde el nivel macro organiza la estructura de cada instancia, discriminando entre elementos que refieren a otros, que son clasificados como *Sección*, de los que no refieren a ningún otro elemento, a los que consolida como *Segmento*.

Al mismo tiempo gestiona la sucesión de propiedades entre referente y referido. Resuelve el conjunto de propiedades resultantes de cada elemento y los dispone consecutivamente para ser consumidos en el nivel de abstracción superior.

#### Complemento

Importa rutinas en el módulo usuario (declarado en `complementos` de la pista) como procesos capaces de manipular propiedades de cualquier elemento de la *Pista*.

Cada *Segmento* busca coincidencia entre cada método de usuario y clave en sus propiedades, delegando a la rutina de usuario la manipulación de los valores de la propiedad subordinada.

#### Unidad

Justifica esta meta clase la noción de una capa superior de agrupamiento y registro. Ademas de ahorrar alguna redundancia, habilita información de capacidad analítica en la salida verbal.

[41] Van Rossum y Drake Jr (1995) 10.3: Command Line Arguments, 5.5: Dictionaries, 3.1.3: Lists, 9: Classes.

[42] El extracto de código correspondiente a cada constructor se encuentran en Apéndice 6.1: Secuencia, Apéndice 6.2: Pista, Apéndice 6.3: Complemento, Apéndice 6.4: Unidad, Apéndice 6.5 : Sección, Apéndice 6.6: Segmento y Apéndice 6.7: Articulación respectivamente.

### Sección

Representa conjuntos de *Secciones* y/o *Segmentos*. Es un grupo de elementos que, sin vincular articulaciones directamente y sin incidir en el discurso resultante, a nivel lógico mantiene relaciones de referencia.

### Segmento

En contra partida, esta clase reúne los mecanismos para preparar y producir *Articulaciones*.

Es responsable de administrar `complementos`, distinguir actualizaciones de valor entre instancias y gestiona alturas, trasponiendo el puntero dentro el set de registración y trasportando la nota resultante.

Completa secuencialmente patrones dispares alineándolos en relación a la lista de mayor extension. Invoca una instancia de *Articulacion* para cada miembro del conjunto resultante de esta combinatoria.

### Articulación

Esta capa corresponde a pronunciamientos en el discurso. Es el producto de mecanismos superiores y la materia prima interpretada en la *Secuencia* final asignando una llamada al codificador por cada instancia.

## 4.3. Demostraciones

### 4.3.1. Melodia Simple

**YAML**

```yaml
nombre: 'Feliz Cumpleaños'
base: &base
  registracion: [
    -12, -10, -8, -7, -5, -3, -1,
      0,   2,  4,  5,  7,  9, 11,
     12,  14, 15, 17, 19, 21, 23,
     24
  ]
  transportar: 60
  duraciones: [ 1.5 ]
  metro: 3/4
forma: [ 'espera', 'estrofa', 'estribo' ]
unidades:
  espera:
    <<: *base
    dinamicas: [ 0 ]
    duraciones: [ 2 ]
  a: &a
```

```
  <<: *base
  alturas: [ 5, 5, 6, 5 ]
  duraciones: [ .75, .25, 1, 1 ]
  letras: [ 'Que', 'los', 'cum', 'plas,' ]
b: &b
  <<: *base
  alturas: [ 8, 7 ]
  duraciones: [ 1, 2 ]
  letras: [ 'fe', 'liz.' ]
b`:
  <<: *b
  transponer: 1
a`:
  <<: *a
  alturas: [ 5, 5, 12, 10 ]
a``:
  <<: *a
  alturas: [ 8, 8, 7, 6 ]
  letras: [ 'que', 'los', 'cum', 'plas,' ]
a```:
  <<: *a
  alturas: [ 11, 11, 10, 8 ]
  letras: [ 'que', 'los', 'cum', 'plas,' ]
A:
  forma: [ 'a', 'b' ]
A`:
  forma: [ 'a' , 'b`' ]
B:
  forma: [ 'a`', 'a``' ]
A``:
  forma: [ 'a```', 'b`' ]
estrofa:
  forma: [ 'A', 'A`' ]
estribo:
  forma: [ 'B', 'A``' ]
```

---

### Partitura

### Salida detallada

```
PISTA 0: Feliz Cumpleaños
       id #  LVL RCR
SGMT0  0  0  0   0   +espera --
SECC0  1  0  0   0   +estrofa =
SECC1  2  1  1   0   +-A ======
SGMT1  3  1  2   0   +--a -----
SGMT2  4  2  2   0   +--b -----
```

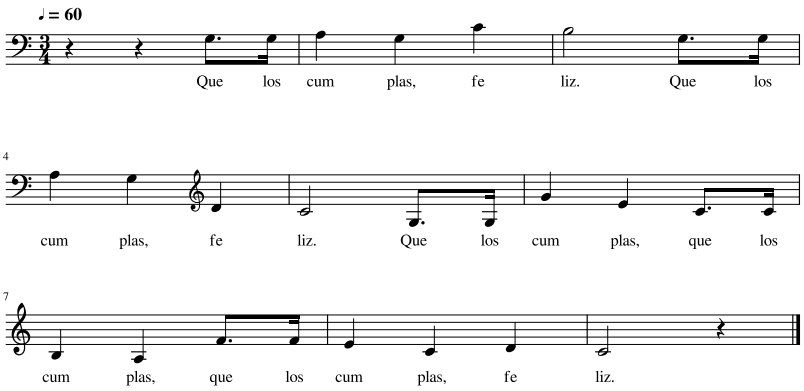


Figura 1: Feliz Cumpleaños

```
SECC2  5  2  1   0   +-A` =====
SGMT3  6  3  2   1   +--a -----
SGMT4  7  4  2   0   +--b` ----
SECC3  8  3  0   0   +estribo =
SECC4  9  4  1   0   +-B ======
SGMT5  10 5  2   0   +--a` ----
SGMT6  11 6  2   0   +--a`` ---
SECC5  12 5  1   0   +-A`` ====
SGMT7  13 7  2   0   +--a``` --
SGMT8  14 8  2   1   +--b` ----
```

### 4.3.2. Multiples Pistas

### YAML

---

```yaml
nombre: 'Billie Jean: Bass'
base: &base
  registracion: [
    -12, -10, -9, -7, -5, -3, -2,
      0,   2,  3,  5,  7,  9, 10,
     12,  14, 15, 17, 19, 21, 22,
     24
  ]
  transportar: 35 #B
  BPMs: [ 100 ]
  duraciones: [ .5 ]
  metro: 4/4
  dinamicas: [ 0.5 ]
  canal:  1
  programas: [ 34 ]
unidades:
```

```
  a:
    <<: *base
    alturas: [ 8, 5, 7, 8 ]
  b:
    <<: *base
    alturas: [ 7, 5, 4, 5 ]
  vuelta:
    forma: [ 'a', 'b' ]
    reiterar: 2
forma: [ 'vuelta' ]
```

---

---

```
nombre: 'Billie Jean: Drum'
base: &base
  registracion: [ 36,38,44 ]
  transportar: 0
  duraciones: [ .5 ]
  dinamicas: [ .5 ]
  BPMs: [ 100 ]
  canal: 9
unidades:
  a:
    <<: *base
    #alturas: [ 1,2,3 ]
    alturas: [ 3 ]
    voces:
      - [ 0 ]
      - [ -2 ]
  b:
    <<: *base
    alturas: [ 3 ]
  c:
    <<: *base
    alturas: [ 3 ]
    voces:
      - [ 0 ]
      - [ -2 ]
      - [ -1 ]
  vuelta:
    forma: [ 'a', 'b', 'c','b' ]
    reiterar: 4
forma: [ 'vuelta' ]
```

---

---

```
nombre: 'Billie Jean: Keys'
base: &base
  registracion: [
      0,   2,  3,  5,  7,  9, 10,
```

```
    12,  14, 15, 17, 19, 21, 22,
    24,  26, 27, 29, 31, 35, 38,
    46
  ]
  alturas: [ 1 ]
  transportar: 59 #B
  duraciones: [ 1.5, 2.5 ]
  BPMs: [ 100 ]
  dinamicas: [ .7 ]
  canal: 0
  voces:
    - [ 4 ]
    - [ 2 ]
    - [ 0 ]
unidades:
  a:
    <<: *base
    alturas: [ 1, 2 ]
  b:
    <<: *base
    alturas: [ 3, 2 ]
  vuelta:
    forma: [ 'a', 'b' ]
forma: [ 'vuelta' ]
```

---

### Partitura

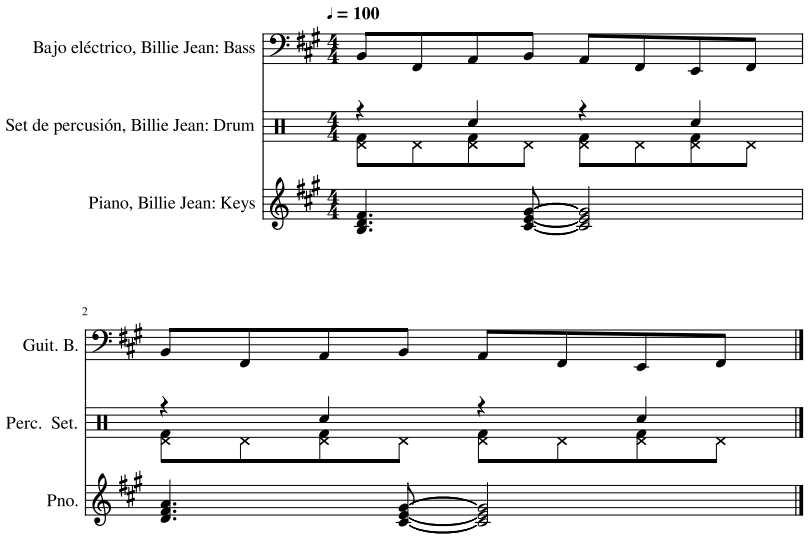


Figura 2: Billie Jean

**Salida detallada**

```
PISTA 0: Billie Jean: Bass
       id #  LVL RCR
SECC0  0  0  0   0   +vuelta ==
SGMT0  1  0  1   0   +-a ------
SGMT1  2  1  1   0   +-b ------
SECC1  3  1  0   1   +vuelta ==
SGMT2  4  2  1   1   +-a ------
SGMT3  5  3  1   1   +-b ------

PISTA 1: Billie Jean: Drum
       id #  LVL RCR
SECC2  6  0  0   0   +vuelta ==
SGMT4  7  0  1   0   +-a ------
SGMT5  8  1  1   0   +-b ------
SGMT6  9  2  1   0   +-c ------
SGMT7  10 3  1   1   +-b ------
SECC3  11 1  0   1   +vuelta ==
SGMT8  12 4  1   1   +-a ------
SGMT9  13 5  1   2   +-b ------
SGMT10 14 6  1   1   +-c ------
SGMT11 15 7  1   3   +-b ------
SECC4  16 2  0   2   +vuelta ==
SGMT12 17 8  1   2   +-a ------
SGMT13 18 9  1   4   +-b ------
SGMT14 19 10 1   2   +-c ------
SGMT15 20 11 1   5   +-b ------
SECC5  21 3  0   3   +vuelta ==
SGMT16 22 12 1   3   +-a ------
SGMT17 23 13 1   6   +-b ------
SGMT18 24 14 1   3   +-c ------
SGMT19 25 15 1   7   +-b ------

PISTA 2: Billie Jean: Keys
       id #  LVL RCR
SECC6  26 0  0   0   +vuelta ==
SGMT20 27 0  1   0   +-a ------
SGMT21 28 1  1   0   +-b ------
```

# 5. Conclusiones

Lidiar con un sistema de producción musical carente de una representación gráfica, sumando a la incapacidad de manipulación mediante interfaces de entrada espaciales aportan un grado de abstracción que puede provocar cierta reticencia,

válida y reconocida previamente en el estudio del marco referencial, a la cual la aplicación de este proyecto tampoco está exenta.

Esta oposición se ve compensada por la promesa de ventajas que, si bien de carácter general fueron reseñadas anteriormente, en un escenario especifico a la producción, manipulación y archivo de información musical, se destaca la capacidad de acceso a elementos a través de la estructura de relaciones de referencia, que propaga actualizaciones en todo el discurso, agilizando al proceso compositivo desde la planificación de obra.

La secuenciación alcanza considerable nivel de detalle en el control y sincronía entre eventos mediante procedimientos relativamente sencillos. El registro aislado de partes habilita su convivencia en diferentes piezas.

La adopción de un proceder similar al propuesto promueve el desarrollo de un corpus de información analítica preparada para tareas de musicología y trabajo teórico.

Virtudes que no se atribuye fomentar a entornos basados en interfaces gráficas de usuario.

Potenciales aplicaciones y desarrollos derivados pueden ser:

- La construcción de un espacio virtual de trabajo colaborativo en tiempo real y a distancia, mediante el acceso en linea a una instalación publica.
- Es fácil proyectar un entorno de improvisación musical enviando la salida a registros para reiterar la lectura/escritura de actualizaciones en tiempo real.
- Con ciertas consideraciones es factible la confección de modelos constringir precisión necesaria para constringir el entrenamiento de agentes informáticos autónomos basados en redes neuronales.

# 6. Apéndice

## 6.1. Secuencia

```
from .pista import Pista
class Secuencia:

  """ Consolida Pistas.
  Recorre articulaciones y cambios de parámetros entre los elementos.
  Reúne todos los pronunciamientos en único flujo de instrucciones.
  """

  def __init__(
      self,
      definiciones,
      verbose,
```

```
      copy
    ):
    self.definiciones = definiciones
    self.pistas = []
    self.verbose = verbose
    self.copyright = copy


    for d in self.definiciones:
      r = {
        **Pista.defactos,
        **d
      }
      pista = Pista(
        nombre    = r[ 'nombre' ],
        paleta    = r[ 'unidades' ],
        forma     = r[ 'forma' ],
        plugin    = r[ 'complementos' ],
        secuencia = self,
      )
      self.pistas.append( pista )

  @property
  def llamadas( self ):
    """ A partir de cada definicion agrega una Pista. """
    llamadas = []
    for pista in self.pistas:

      if self.verbose:
        print( pista.verbose( self.verbose ) )

      """ Generar track p/c pista """
      delta = 0
      track = pista.numero

      """ Parametros de Pista Primer articulación de la parte, agregar
      eventos fundamentales: pulso, armadura de clave, compás y programa.
      """
      llamadas.append([
        'addTrackName',
        track,
        delta,
        pista.nombre
      ])

      if self.copyright:
```

```
59          llamadas.append([
60            'addCopyright',
61            track,
62            delta,
63            self.copyright
64          ])
65
66        """ Loop principal:
67        Genera una secuencia de eventos MIDI lista de articulaciones.  """
68
69        for segmento in pista.segmentos:
70          canal = segmento.canal
71          #delta += segmento.desplazar
72
73          if delta < 0:
74           raise ValueError( 'No se puede desplazar antes q el inicio' )
75           pass
76
77          """ Agregar propiedades de segmento. """
78
79          if segmento.cambia( 'metro' ):
80            llamadas.append([
81              'addTimeSignature',
82              track,
83              delta,
84              segmento.metro[ 'numerador' ],
85              segmento.metro[ 'denominador' ],
86              segmento.metro[ 'relojes_por_tick' ],
87              segmento.metro[ 'notas_por_pulso' ]
88            ])
89
90          if segmento.cambia( 'bpm' ):
91            llamadas.append([
92              'addTempo',
93              track,
94              delta,
95              segmento.bpm,
96            ])
97
98          if segmento.cambia( 'clave' ):
99            llamadas.append([
100             'addKeySignature',
101             track,
102             delta,
103             segmento.clave[ 'alteraciones' ],
104             1, # multiplica por el n de alteraciones
```

```
105             segmento.clave[ 'modo' ]
106           ])
107
108         if segmento.afinacionNota:
109           llamadas.append([
110            'changeNoteTuning',
111             track,
112             segmento.afinacionNota[ 'afinaciones' ],
113             segmento.afinacionNota[ 'canalSysEx' ],
114             segmento.afinacionNota[ 'tiempoReal' ],
115             segmento.afinacionNota[ 'programa' ],
116           ])
117
118         if segmento.afinacionBanco:
119           llamadas.append([
120             'changeTuningBank',
121             track,
122             canal,
123             delta,
124             segmento.afinacionBanco[ 'banco' ],
125             segmento.afinacionBanco[ 'ordenar' ],
126           ])
127
128         if segmento.afinacionPrograma:
129           llamadas.append([
130             'changeTuningProgram',
131             track,
132             canal,
133             delta,
134             segmento.afinacionPrograma[ 'programa' ],
135             segmento.afinacionPrograma[ 'ordenar' ],
136           ])
137
138         if segmento.sysEx:
139           llamadas.append([
140            'addSysEx',
141             track,
142             delta,
143             segmento.sysEx[ 'fabricante' ],
144             segmento.sysEx[ 'playload' ],
145           ])
146
147         if segmento.uniSysEx:
148           llamadas.append([
149            'addUniversalSysEx',
150             track,
```

```
151             delta,
152             segmento.uniSysEx[ 'codigo' ],
153             segmento.uniSysEx[ 'subCodigo' ],
154             segmento.uniSysEx[ 'playload' ],
155             segmento.uniSysEx[ 'canal' ],
156             segmento.uniSysEx[ 'tiempoReal' ],
157           ])
158
159         if segmento.NRPN:
160           llamadas.append([
161            'makeNRPNCall',
162             track,
163             canal,
164             delta,
165             segmento.NRPN[ 'control_msb' ],
166             segmento.NRPN[ 'control_lsb' ],
167             segmento.NRPN[ 'data_msb' ],
168             segmento.NRPN[ 'data_lsb' ],
169             segmento.NRPN[ 'ordenar' ],
170           ])
171
172         if segmento.RPN:
173           llamadas.append([
174            'makeRPNCall',
175             track,
176             canal,
177             delta,
178             segmento.RPN[ 'control_msb' ],
179             segmento.RPN[ 'control_lsb' ],
180             segmento.RPN[ 'data_msb' ],
181             segmento.RPN[ 'data_lsb' ],
182             segmento.RPN[ 'ordenar' ],
183           ])
184
185         for articulacion in segmento.articulaciones:
186           """ Agrega cualquier cambio de parametro,
187           comparar cada uno con la articulacion previa. """
188
189           if articulacion.cambia( 'bpm' ):
190             llamadas.append([
191               'addTempo',
192               track,
193               delta,
194               articulacion.bpm,
195             ])
196
```

```
197             if articulacion.cambia( 'programa' ):
198               llamadas.append([
199                  'addProgramChange',
200                  track,
201                  canal,
202                  delta,
203                  articulacion.programa
204               ])
205
206             if articulacion.letra:
207               llamadas.append([
208                'addText',
209                 track,
210                 delta,
211                 articulacion.letra
212               ])
213
214             if articulacion.tono:
215               llamadas.append([
216                  'addPitchWheelEvent',
217                  track,
218                  canal,
219                  delta,
220                  articulacion.tono
221               ])
222
223             """ Agregar nota/s (altura, duracion, dinamica).
224             Si existe acorde en la articulación armar una lista con cada voz
225             superpuesta.  o una lista de solamente un elemento.  """
226             voces = [ articulacion.altura ]
227             if articulacion.acorde:
228               voces = articulacion.acorde
229
230             for voz in voces:
231               if articulacion.dinamica:
232                 llamadas.append([
233                   'addNote',
234                   track,
235                   canal,
236                   voz,
237                   delta,
238                   articulacion.duracion,
239                   articulacion.dinamica
240                 ])
241               #else:
242               #  print('eo')
```

```
243
244           if articulacion.controles:
245             """ Agregar cambios de control """
246             for control in articulacion.controles:
247               for control, valor in control.items():
248                 llamadas.append([
249                  'addControllerEvent',
250                   track,
251                   canal,
252                   delta,
253                   control,
254                   valor,
255                 ])
256
257           delta += articulacion.duracion
258     return llamadas
```

## 6.2. Pista

```
1  import os
2  from argumentos import Excepcion
3  from .complemento import Complemento
4  from .seccion  import Seccion
5  from .segmento import Segmento
6
7  class Pista:
8
9    """ Organiza la estructura de cada instancia desde el nivel macro.
10   Clasifica:
11     Secciones (elementos que refieren a otros)
12     Segmento (no refieren a ningún otro elemento)
13   Gestiona la sucesión de propiedades entre referente y referido.
14   Resuelve el conjunto de propiedades resultantes de cada elemento y los dispone
15   consecutivamente para ser consumidos en el nivel de abstracción superior.
16   """
17
18   cantidad = 0
19
20   defactos = {
21     'nombre'       : '',
22     'unidades'     : [ None ],
23     'forma'        : [ None ],
24     'complementos' : None,
25   }
```

```
26
27  def __str__( self ):
28    o  = 'PISTA ' + str( self.numero) + ': '+ str( self.nombre  )
29    return o
30
31  def verbose( self, verbosidad = 0 ):
32    if verbosidad > 0:
33      o = str( self ) + ' '
34      o = str( self ) + ' '
35      if verbosidad > 1:
36        o += '\n       id #  LVL RCR\n'
37        for e in self.elementos:
38          o += e.verbose( verbosidad )
39          o += '\n'
40      return o
41
42  def __init__(
43    self,
44    nombre,
45    paleta,
46    forma,
47    plugin,
48    secuencia,
49  ):
50    self.nombre     = nombre
51    self.paleta     = paleta
52    self.forma      = forma
53    self.secuencia  = secuencia
54    self.plugin     = plugin
55    self.numero     = Pista.cantidad
56    Pista.cantidad += 1
57
58    self.secciones  = []
59    self.segmentos  = []
60    self.seccionar( self.forma )
61
62  @property
63  def complemento( self ):
64    ruta = self.plugin
65    c = None
66    if ruta and os.path.exists( ruta ):
67      # TODO Tirar excepción
68      # and p not in Complemento.registro:
69      Complemento.registro.append( ruta )
70      c = Complemento( ruta )
71    return c
```

```
72
73    @property
74    def elementos( self ):
75      return sorted(
76        self.secciones + self.segmentos,
77        key = lambda x: x.numero
78      )
79
80    @property
81    def tiempo( self ):
82      # duracion en segundos
83      return sum( [ s.tiempo for s in self.segmentos ] )
84
85    """ Organiza unidades según relacion de referencia """
86    def seccionar(
87      self,
88      forma = None,
89      nivel = 0,
90      herencia = {},
91      referente = None,
92    ):
93      nivel += 1
94      """ Limpiar parametros q no se heredan.  """
95      herencia.pop( 'forma', None )
96      herencia.pop( 'reiterar', None )
97
98      for unidad in forma:
99        try:
100         if unidad not in self.paleta:
101           error = "PISTA: \"" + self.nombre + "\""
102           error += " NO ENCUENTRO: \"" + unidad + "\"  "
103           raise Excepcion( unidad, error )
104           pass
105         original = self.paleta[ unidad ]
106         sucesion = {
107           **original,
108           **herencia,
109         }
110         reiterar = 1
111         if 'reiterar' in original:
112           reiterar = original[ 'reiterar' ]
113         for r in range( reiterar ):
114           if 'forma' not in original:
115             segmento = Segmento(
116               pista       = self,
117               nombre      = unidad,
```

```
118               nivel       = nivel - 1,
119               orden       = len( self.segmentos ),
120               recurrencia = sum(
121                 [ 1 for e in self.segmentos if e.nombre == unidad ]
122               ),
123               referente = referente,
124               propiedades = sucesion,
125             )
126             self.segmentos.append( segmento )
127           else:
128             seccion = Seccion(
129               pista       = self.nombre,
130               nombre      = unidad,
131               nivel       = nivel - 1,
132               orden       = len( self.secciones ),
133               recurrencia = sum(
134                 [ 1 for e in self.secciones if e.nombre == unidad ]
135               ),
136               referente   = referente,
137             )
138             seccion.referidos = original['forma']
139             self.secciones.append( seccion )
140             elemento = seccion
141             self.seccionar(
142               original[ 'forma' ],
143               nivel,
144               sucesion,
145               seccion,
146             )
147       except Excepcion as e:
148           print( e )
```

## 6.3. Complemento

```
1  import importlib.util as importar
2
3  class Complemento:
4
5      """ Interfaz para complementos de usuario.
6      Declarar ubicación del paquete en propiedades de track.
7      complementos: 'enchufes.py'
8      En propiedades de segmento invocar metodo y subscribir, propiedad y
9      argumentos.
10     metodo: # en: enchufes.py
```

```
      propiedad_a_manipular1: argumentos
      propiedad_a_manipular2: argumentos
    fluctuar:
      dinamicas: .5
      alturas: 2
    """

    cantidad = 0
    registro = []

    def __str__(
      ubicacion,
    ):
      return self.nombre

    def __init__(
      self,
      ruta
    ):
     self.ruta = ruta
     self.nombre = ruta.split( '.' )[ 0 ]
     Complemento.cantidad += 1
     spec = importar.spec_from_file_location(
       self.nombre,
       self.ruta
     )
     if spec:
       modulo = importar.module_from_spec( spec )
       spec.loader.exec_module( modulo )
       self.modulo = modulo
```

## 6.4. Unidad

```
class Unidad():

  """ Meta clase: Secciones y Segmentos
  Agrupamiento y registro
  """

  cantidad = 0
  def __str__( self ):
    o  = str( self.numero ) + ' '
    if(self.numero < 10 ):
      o += ' '
```

```
    o += str( self.orden ) + ' '
    if(self.orden < 10 ):
      o += ' '
    o += str( self.nivel )  + '  '
    if(self.nivel < 10 ):
      o += ' '
    o += str( self.recurrencia ) + ' '
    if(self.recurrencia < 10 ):
      o += ' '
    o += ' +' + str( '-' * ( self.nivel ) )
    o += self.nombre
    return o

  def __init__(
    self,
    pista,
    nombre,
    nivel,
    orden,
    recurrencia,
    referente,
  ):
    self.pista  = pista
    self.nombre = nombre
    self.nivel = nivel
    self.orden = orden
    self.recurrencia = recurrencia
    self.referente = referente
    self.numero = Unidad.cantidad
    Unidad.cantidad += 1
```

## 6.5. Sección

```
from .unidad import Unidad

class Seccion(  Unidad ):

  """ Conjuntos de Secciones y/o Segmentos.
  Conserva relaciones de referencia a nivel lógico
  """
  cantidad = 0

  def verbose( self, vebosidad = 0 ):
    #o =  ('=' * 70 ) + '\n'
```

```
    o = self.tipo + ''
    o += str( self.numero_seccion ) + ' '
    if(self.numero_seccion < 10 ):
      o += ' '
    o += str( self ) + ' '
    o += '=' * ( 8 - ( len( self.nombre ) + self.nivel ) )
    return o

  def __init__(
    self,
    pista,
    nombre,
    nivel,
    orden,
    recurrencia,
    referente
  ):
    Unidad.__init__(
      self,
      pista,
      nombre,
      nivel,
      orden,
      recurrencia,
      referente
    )
    self.numero_seccion = Seccion.cantidad
    Seccion.cantidad += 1
    self.nivel = nivel
    self.tipo = 'SECC'
```

## 6.6. Segmento

```
import math
from .unidad import Unidad
from .articulacion import Articulacion
from .complemento import Complemento


class Segmento( Unidad ):

  """ Mecanismos para preparar y producir Articulaciones. """

  cantidad = 0
```

```
12
13  defactos = {
14
15    # Propiedades de Segmento
16    'canal'    : 0,
17    'revertir' : None,
18    'NRPN'     : None,
19    'RPN'      : None,
20
21    # Props. que NO refieren a Meta Canal
22    'metro'             : '4/4',
23    'alteraciones'      : 0,
24    'modo'              : 0,
25    'afinacionNota'     : None,
26    'afinacionBanco'    : None,
27    'afinacionPrograma' : None,
28    'sysEx'             : None,
29    'uniSysEx'          : None,
30
31    # Procesos de Segmento
32    'transportar' : 0,
33    'transponer'  : 0,
34    'reiterar'    : 1,
35
36    # Propiedades de Articulacion
37    'BPMs'         : [ 60 ],
38    'programas'    : [ None ],
39    'duraciones'   : [ 1 ],
40    'dinamicas'    : [ 1 ],
41    'registracion' : [ 1 ],
42    'alturas'      : [ 1 ],
43    'letras'       : [ None ],
44    'tonos'        : [ 0 ],
45    'voces'        : None,
46    'controles'    : None,
47
48  }
49
50  def verbose( self, vebosidad = 0 ):
51    o = self.tipo + ''
52    o += str( self.numero_segmento) + ' '
53    if(self.numero_segmento < 10 ):
54      o += ' '
55    o += str( self ) + ' '
56    o += '-' * (8 - (len( self.nombre ) + self.nivel))
57    if vebosidad > 2:
```

```
58        o +=  '\n' + ('.' * 70 )
59        o += '\n\tORD\tBPM\tDUR\tDIN\tALT\tLTR\tTON\tCTR\n'
60        for a in self.articulaciones:
61          o += str( a )
62        o +=  ('.' * 70 )
63      return o
64
65    def __init__(
66      self,
67      pista,
68      nombre,
69      nivel,
70      orden,
71      recurrencia,
72      referente,
73      propiedades
74    ):
75      Unidad.__init__(
76        self,
77        pista,
78        nombre,
79        nivel,
80        orden,
81        recurrencia,
82        referente
83      )
84      self.numero_segmento = Segmento.cantidad
85      Segmento.cantidad += 1
86      self.tipo = 'SGMT'
87
88      self.props = {
89        **Segmento.defactos,
90        **propiedades
91      }
92
93      """ Preparar Segmento """
94
95      # """ Cambia el sentido de los parámetros de
96      # articulación """
97      # self.revertir = self.props[ 'revertir' ]
98      # if self.revertir:
99      #   if isinstance( self.revertir , list ):
100     #     for r in self.revertir:
101     #       if r in self.props:
102     #         self.props[ r ].reverse()
103     #   elif isinstance( self.revertir , str ):
```

```
104     #     if revertir in self.props:
105     #       self.props[ self.revertir ].reverse()
106
107     self.canal             = self.props[ 'canal' ]
108     self.reiterar          = self.props[ 'reiterar' ]
109     self.transponer        = self.props[ 'transponer' ]
110     self.transportar       = self.props[ 'transportar' ]
111     self.alteraciones      = self.props[ 'alteraciones' ]
112     self.modo              = self.props[ 'modo' ]
113     self.afinacionNota     = self.props[ 'afinacionNota' ]
114     self.afinacionBanco    = self.props[ 'afinacionBanco' ]
115     self.afinacionPrograma = self.props[ 'afinacionPrograma' ]
116     self.sysEx             = self.props[ 'sysEx' ]
117     self.uniSysEx          = self.props[ 'uniSysEx' ]
118     self.NRPN              = self.props[ 'NRPN' ]
119     self.RPN               = self.props[ 'RPN' ]
120     self.registracion      = self.props[ 'registracion' ]
121
122     self.programas         = self.props[ 'programas' ]
123     self.duraciones        = self.props[ 'duraciones' ]
124     self.BPMs              = self.props[ 'BPMs' ]
125     self.dinamicas         = self.props[ 'dinamicas' ]
126     self.alturas           = self.props[ 'alturas' ]
127     self.letras            = self.props[ 'letras' ]
128     self.tonos             = self.props[ 'tonos' ]
129     self.voces             = self.props[ 'voces' ]
130     self.capas             = self.props[ 'controles' ]
131
132     self.bpm = self.BPMs[0]
133     self.programa = self.programas[0]
134
135     """ COMPLEMENTOS
136         Pasar propiedades por método de usuario
137     """
138     if self.pista.complemento:
139       for metodo in dir( self.pista.complemento.modulo ):
140         if metodo in self.props:
141           for clave in self.props[ metodo ]:
142             original = getattr( self, clave )
143             argumentos = self.props[ metodo ][ clave ]
144             # print( metodo, ':', clave, argumentos )
145             modificado = getattr(
146                self.pista.complemento.modulo,
147                metodo,
148             )( original, argumentos )
149             setattr( self, clave, modificado )
```

```
150
151   @property
152   def precedente( self ):
153     n = self.orden
154     o = self.pista.segmentos[ n - 1 ]
155     return o
156
157   def obtener( self, key ):
158       try:
159         o = getattr( self, key )
160         return o
161       except AttributeError as e:
162         return e
163
164   def cambia(
165       self,
166       key
167     ):
168     este = self.obtener( key )
169     anterior = self.precedente.obtener( key )
170     if (
171       self.orden == 0
172       and este
173     ):
174       return True
175     return anterior != este
176
177   @property
178   def tiempo( self ):
179     # duración en segundos
180     return sum( [ a.tiempo for a in self.articulaciones ] )
181
182   @property
183   def metro( self ):
184     metro = self.props[ 'metro' ].split( '/' )
185     denominador = int(
186       math.log10( int( metro[ 1 ] ) ) / math.log10( 2 )
187     )
188     return {
189       'numerador'        : int( metro[ 0 ] ),
190       'denominador'      : denominador,
191       'relojes_por_tick' : 12 * denominador,
192       'notas_por_pulso'  : 8,
193     }
194
195   @property
```

```
196   def clave( self ):
197     return {
198       'alteraciones' : self.alteraciones,
199       'modo' : self.modo
200     }
201
202   @property
203   def ganador( self ):
204     """ Evaluar que propiedad lista es el que mas valores tiene.  """
205     self.ganador_voces = [ 0 ]
206     if self.voces:
207       self.ganador_voces = max( self.voces, key = len )
208     self.ganador_capas = [ 0 ]
209     if self.capas:
210       self.ganador_capas = max( self.capas , key = len )
211
212     candidatos = [
213       self.dinamicas,
214       self.duraciones,
215       self.alturas,
216       self.letras,
217       self.tonos,
218       self.BPMs,
219       self.programas,
220       self.ganador_voces,
221       self.ganador_capas,
222     ]
223     return max( candidatos, key = len )
224
225   @property
226   def cantidad_pasos( self ):
227     return len( self.ganador )
228
229   @property
230   def articulaciones( self ):
231     """ Consolidar "Articulaciones"
232     combinar parámetros: altura, duración, dinámica, etc. """
233     o = []
234     for paso in range( self.cantidad_pasos ):
235       """ Alturas, voz y superposición voces. """
236       altura = self.alturas[ paso % len( self.alturas ) ]
237       acorde = []
238       #nota   = altura
239       """ Trasponer puntero dentro el set de registración.
240         transportar la nota resultante.
241       """
```

```
242       n = self.registracion[
243         ( ( altura - 1 ) + self.transponer ) % len( self.registracion )
244       ]
245       nota = self.transportar + n
246       """ Armar superposicion de voces. """
247       if self.voces:
248         for v in self.voces:
249           voz = (
250             altura  + ( v[ paso % len( v ) ] )
251           ) + self.transponer
252           acorde += [
253             self.transportar +
254             self.registracion[ (voz - 1) % len( self.registracion ) ]
255           ]
256       """ Cambios de control. """
257       controles = []
258       if self.capas:
259         for capa in self.capas:
260           controles += [ capa[ paso % len( capa ) ] ]
261       """ Articulación a secuenciar. """
262       articulacion = Articulacion(
263          segmento  = self,
264          orden     = paso,
265          bpm       = self.BPMs[ paso % len( self.BPMs ) ],
266          programa  = self.programas[ paso % len( self.programas ) ],
267          # TODO advertir si lista de duraciones vacias
268          duracion  = self.duraciones[ paso % len( self.duraciones ) ],
269          dinamica  = self.dinamicas[ paso % len( self.dinamicas ) ],
270          nota      = nota,
271          acorde    = acorde,
272          tono      = self.tonos[ paso % len( self.tonos ) ],
273          letra     = self.letras[ paso % len( self.letras ) ],
274          controles = controles,
275       )
276       o.append( articulacion )
277     return o
278
```

## 6.7. Articulación

```
1 class Articulacion:
2
3   """ Pronunciamientos. """
4
```

```
5   cantidad = 0
6
7   def __str__( self ):
8     o  = 'ART' + str( self.numero ) + '\t'
9     o += str( self.orden ) + '\t'
10    o += str( self.bpm ) + '\t'
11    o += str( round(self.duracion, 2) ) + '\t'
12    o += str( self.dinamica ) + '\t'
13    o += str( self.altura ) + '\t'
14    o += str( self.letra ) + '\t'
15    #o += str( self.tono) + '\t'
16    o += str( self.controles) + '\n'
17    return o
18
19  def __init__(
20    self,
21    segmento,
22    orden,
23    bpm,
24    programa,
25    duracion,
26    dinamica,
27    nota,
28    acorde,
29    tono,
30    letra,
31    controles,
32  ):
33    self.numero = Articulacion.cantidad
34    Articulacion.cantidad += 1
35
36    self.segmento  = segmento
37    self.orden     = orden
38    self.bpm       = bpm
39    self.programa  = programa
40    self.tono      = tono
41    self._dinamica  = dinamica
42    self.duracion  = duracion
43    self.controles = controles
44    self.altura    = nota
45    self.letra     = letra
46    self.acorde    = acorde
47
48  @property
49  def precedente( self ):
50    n = self.orden
```

```
51      o = self.segmento.articulaciones[ n - 1]
52      if  n == 0:
53        o = self.segmento.precedente.articulaciones[ - 1 ]
54      return o
55
56    def obtener( self, key ):
57        try:
58          o = getattr( self, key )
59          return o
60        except AttributeError as e:
61          return e
62
63    def cambia( self, key ):
64        este = self.obtener( key )
65        anterior = self.precedente.obtener( key )
66        if (
67          self.segmento.orden == 0
68          and self.orden == 0
69          and este
70        ):
71          return True
72        return anterior != este
73
74    @property
75    def relacion( self ):
76        return 60 / self.bpm
77
78    @property
79    def tiempo( self ):
80      # duracion en segundos
81      return self.duracion * self.relacion
82
83    @property
84    def dinamica( self ):
85      viejo_valor = self._dinamica
86      viejo_min = 0
87      viejo_max = 1
88      nuevo_min = 0
89      nuevo_max = 126
90      nuevo_valor = (
91        ( viejo_valor - viejo_min ) / ( viejo_max - viejo_min )
92      ) * ( nuevo_max - nuevo_min) + nuevo_min
93      return int( min( max( nuevo_valor, nuevo_min ), nuevo_max ) )
```

# Bibliografía


BEN-KIKI, O., EVANS, C. y INGERSON, B., 2005. Yaml ain't markup language (yaml™) version 1.1. *yaml. org, Tech. Rep*, pp. 23.

BEN-KIKI OREN, E.C. y INGY, 2009. YAML Version 1.2 Specification. [en línea]. Disponible en: http://yaml.org/spec/1.2/spec.html.

CLARK, C. y TINDALE, A., 2014. Flocking: A Framework for Declarative Music-Making on the Web. *The Joint Proceedings of the ICMC and SMC*, vol. 1, no. 1, pp. 50-57.

COOMBS, J.H., RENEAR, A.H. y DEROSE, S.J., 1987. Markup Systems and the Future of Scholarly Text Processing. *Commun. ACM* [en línea], vol. 30, no. 11, pp. 933-947. ISSN 0001-0782. DOI 10.1145/32206.32209. Disponible en: http://doi.acm.org/10.1145/32206.32209.

GOOD, M., 2001. MusicXML: An Internet-Friendly Format for Sheet Music. *Proceedings of XML* [en línea], Disponible en: http://michaelgood.info/publications/music/musicxml-an-internet-friendly-format-for-sheet-music/.

GRAHAM, P., 2001. *Beating the Averages* [en línea]. 2001. Estados Unidos: Franz Developer Symposium; www.paulgraham.com. Disponible en: http://www.paulgraham.com/avg.html.

GRELA, D., 1992. *Análisis Musical: Una Propuesta Metodológica*. 1992. Rosario, Santa Fe, Argentina: Facultad de Humanidades y Artes. SERIE 5: La música en el Tiempo. Nº1.

HAUS, G. y LUDOVICO, L., 2007. Music Representation of Score, Sound, MIDI, Structure and Metadata All Integrated in a Single Multilayer Environment Based on XML.. S.l.: s.n.,

HUNT, A. y THOMAS, D., 1999. *The Pragmatic Programmer: From Journeyman to Master*. S.l.: The Pragmatic Bookshelf. ISBN 9780201616224.

HUNTLEY, G., 2019. Interprete de Comandos. [en línea]. Disponible en: https://noyaml.com/.

LEEK, J., 2017. The future of education is plain text. [en línea]. Disponible en: https://simplystatistics.org/2017/06/13/the-future-of-education-is-plain-text.

(MMA), M.M.A., 1996. Standard MIDI Files (SMF) Specification. [en línea]. Disponible en: https://www.midi.org/specifications-old/item/standard-midi-files-smf.

MOOLENAAR, B., 2000. Seven habits of effective text editing. [en línea]. Disponible en: http://moolenaar.net/habits.html.

(N.D.), 2018a. PyYAML is a full-featured YAML framework for the Python programming language. [en línea]. Disponible en: https://pyyaml.org/.

(N.D.), 2018b. The Pyhton Standar Library. [en línea]. Disponible en: https://docs.python.org/3/library/index.html.

(N.D.), 2019a. Lenguaje específico de dominio. [en línea]. Disponible en: https://es.wikipedia.org/wiki/Lenguaje_espec%C3%ADfico_de_dominio.

(N.D.), 2019b. YAML Test Matrix. [en línea]. Disponible en: https://matrix.yaml.io/valid.html.

OUALLINE, S., 2001. *Vi iMproved*. S.l.: New Riders Publishing.

PENFOLD, R.A., 1992. *Advanced MIDI Users Guide*. United Kingdom: PC Publishing. ISBN 978-1870775397.

POPE, S.T., 1986. Music Notations and the Representation of Musical Structure and Knowledge. *Perspectives of New Music* [en línea], vol. 24, no. 2, pp. 156-189. DOI 10.2307/833219. Disponible en: https://www.jstor.org/stable/833219.

RAYMOND, E.S., 1997. *The Cathedral and the Bazaar*. 1997. Estados Unidos: Linux Kongress; O'Reilly Media.

RAYMOND, E.S., 1999. *The Art of UNIX Programming*. Estados Unidos: Addison-Wesley Professional. ISBN 978-0131429017.

ROSSUM, G.V., 2018. Python 3.7. [en línea]. Disponible en: https://docs.python.org/3/.

SELFRIDGE-FIELD, E., 1997. *Beyond MIDI: The Handbok of Musical Codes*. Estados Unidos: The MIT Press. ISBN 9780262193948.

STEYN, J., 2001. Music Markup Language. [en línea]. Disponible en: https://steyn.pro/mml.

TORVALDS, L. y HAMANO, J., 2010. Git: Fast version control system. *URL http://git-scm. com*,

VAN ROSSUM, G. y DRAKE JR, F.L., 1995. *Python tutorial*. S.l.: Centrum voor Wiskunde en Informatica Amsterdam.

VARIEGO, J., 2018. *Composición algorítmica. Matemáticas y ciencias de la computación en la creación musical*. S.l.: Universidad Nacional de Quilmes. ISBN 978-987-558-502-7.

WALL, L., 1999. Perl, the first postmodern computer language. [en línea]. Disponible en: https://www.perl.com/pub/1999/03/pm.html/.

WILD, J., 1996. A Review of the Humdrum Toolkit: UNIX Tools for Musical Research, created by David Huron. *Music Theory Online*, vol. 2, no. 7.

WIRTS, M.C., 2016. MIDIUtil. [en línea]. Disponible en: https://midiutil.readthedocs.io.

YZAGUIRRE, G., 2016. Manifiesto del Laboratorio de Software Libre. [en línea]. Disponible en: https://labsl.multimediales.com.ar/Manifiesto_del_Laboratorio_de_Software_Libre_.html.