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

Abstract

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.

Mayo 2019

Buenos Aires, Argentina

Contendios

1 Resumen

El presente trabajo propone un contexto de producción musical puramente textual.

Son el producto de esta investigación un marco de patrones y relaciones gramaticales que posibilitan la representación textual de información con significado musical; un léxico y una sintaxis que definen estructuras musicales contenidas en ficheros de texto serializado[1] y autodescriptivo.

Acompaña esta propuesta un entorno de herramientas, para interprete de línea de comandos. Siendo otro aporte principal del actual desarrollo esta cadena de procesos que consume información suscrita a dicha representación; derivando esta manipulación en la producción de secuencias de mensajes en el estándar MIDI.

La primera parte de este escrito esta dedicada a justificar el objeto de estudio, presentar los motivos de las interrogantes así como también se enumeran 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 la materia prima de esta serie de procesos y se despliegan el resultado de algunos ejemplos como demostración.

A modo de conclusión se proyectan algunas aplicaciones posibles en diferentes escenarios (online, livecodig) y varias disciplinas (IA, archivología).

En el apéndice se exponen el código de los módulos desarrollados en la implementación.

[1] Coombs, Renear y De Rose (1987)

2 Introducción

En esta sección inaugural se enmarca la investigación, argumentando la constricción principal, el por que de la adopción de un sistema de escritura como contenedor 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 proyectos similares con cierta relevancia a este proyecto.

2.1 Justificación

En este apartado se resumen las características principales por las que el texto es idóneo como propósitos generales.

2.1.1 ¿Por qué Texto Plano?

> "…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 enfrenta el texto plano y legible en contraste a la codificación de datos.[2]

Aprovechar. Potencialmente cualquier herramienta de computo puede operar información almacenada en texto plano.

Mínimo Común Denominador. 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 rudimentales 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.

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

Seguro contra toda 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 con las hayan sido creados.[3]

2.1.2 ¿Por qué Interfaz de Linea de Comandos?

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 interfaz para diferentes aplicaciones. La estructura de las instrucciones para esta interfaz *aplicación - argumento - recurso* (su analogía *verbo - adverbio - sujeto*) persiste para cualquier pieza de software. Dicha recurrencía elimina el ejercicio que significa operar de modo distinto cada aplicación, favoreciendo un accionar semejante 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 mas 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 (guión).

Pipeline y Automatización. La composición flujos de procesos complejos encadenando resultados con trabajos.[4]

Acceso remoto. Mas allá del protocolo en el que se base la negociación local/remoto la interfaz de linea 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]

2.2 Motivación

Este proyecto la necesidad de establecer un contexto y proveer los recursos para un procedimiento sencillo y flexible de elaboración discursos musicales unifi-

[3] Leek (2017)

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

[5] Moolenaar (2000)

cando la planificación de obra con la secuenciación MIDI.

Ademas pretende exponer las ventajas de la Interfaz de Linea de Comandos para operar sistemas informáticos 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.[6][7]

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 cosideró Flocking.

2.3.1 MuseData

La base de datos MuseData[8] es un proyecto y a la vez el sistema de codificación principal del Centro de Investigación Asistida por Computador en Humanidades (CCARH). La base de datos fue creado por Walter Hewlett.

Los archivos MuseData tienen el potencial de existir en múltiples formatos comunes de información. La mayoría de las codificaciones derivadas acomodan sólo algunas de las las características incluidas en el master MuseData de codificaciones. El archivo MuseData está diseñado para soportar aplicaciones de sonido, gráficos y análisis. Los formatos derivados de las codificaciones musicales de MuseData que se distribución son: MIDI1, MIDI+ y Humdrum.

2.3.1.1 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 un formato a otro así como también de una edición a otra.

Los archivos MuseData 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.

Las partes de los archivos MuseData siempre tienen la etiqueta 01 para la primera parte, 02 para la segunda parte de la partitura, etc. Conteniendo varias líneas de música, como dos flautas en una partitura de orquesta, o dos sistemas para música de piano. Archivos para diferentes los movimientos de una

[6] Raymond (1997) Capítulo 11: The Social Context of Open-Source Software (p. 11)
[7] Yzaguirre (2016)
[8] Selfridge-Field (1997)

composición se encuentran en directorios separados que usualmente indican el número de movimiento, p. 01, 02, etc.

La exhaustividad de la información dentro de los archivos varía entre dos niveles que en archivos MuseData llamamos Stage 1 y Stage 2. Sólo los archivos Stage 2 son recomendados para aplicaciones serias.

El primer paso en la entrada de datos (Stage 1) captura información básica como duración y altura de las notas. Por ejemplo, normalmente habría cuatro archivos (Violín 1, Violín 2, Viola, Violonchelo) para cada movimiento de un cuarteto de cuerdas. Si el movimiento del cuarteto comienza en metro binario, cambia a metro ternario, y luego vuelve a binario, cada sección métrica tendrá su propio conjunto de partes. Así habría doce archivos para el movimiento. El segundo paso en la entrada de datos (Stage 2) suministra toda la información que no puede ser capturado de forma fiable desde un teclado electrónico. Esto incluye indicaciones para ritmo, dinámica y articulación.

El juicio humano se aplica en el Stage 2. Así, cuando el movimiento del cuarteto de cuerdas citado anteriormente se convierte a la Stage 2, las tres secciones métricas para cada instrumento capturado desde la entrada del teclado se encadenará en un movimiento cada uno. El movimiento tendrá ahora cuatro archivos de datos (uno para Violín 1, otro para Violín 2, Viola, Violonchelo).

El juicio humano también proporciona correcciones y anotaciones a los datos. Algunos tipos de errores (por ejemplo, medidas incompletas) deben corregirse y así consiguen tener sentido para el usuario. Los asuntos que son más discrecionales (tales como alteraciones opcionales de los ornamentos o accidentes) por lo general no se modifica. Las decisiones discrecionales se anotan en archivos que permiten marcas editoriales.

2.3.1.2 La representación MuseData de información musical

El propósito de la sintaxis MuseData es representar el contenido lógico de una pieza musical de una modo neutral. El código se utiliza actualmente 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 música, 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 nuestra opinión es que tal codificación realmente contendría más información que la que el compositor pretende transmitir.
- No se puede anticipar todos los usos a los cuales podrían darse estos datos, pero se pude estar bastante seguro de 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 realización gráfica de los datos o cómo sonido que estos datos representan debe 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 datos independiente, que puede ser fácilmente ignorado por software de aplicación o despojado enteramente de los datos. MuseData software 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[9] 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 es un conjunto de herramientas de línea de comandos, es el lenguaje de programa agnóstico. Muchos han empleado herramientas de Humdrum en secuencias de comandos más grandes que utilizan PERL, Ruby, Python, Bash, LISP y C++.

2.3.2.1 Representación

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

- Su 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.

[9] Wild (1996)

2.3.2.2 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 explotar plenamente el potencial del sistema.

2.3.2.3 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 de el tiempo requerido para aprender a usar Humdrum será más que compensado por ganancias académicas posteriores.

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 lentos para reconocer el potencial del sistema.

Humdrum u otros sistemas como él ofrecen los recursos para una marcar un paradigma para 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 benefician de una pequeña intuición asistida por la máquina es probable que 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 parecer intimidantes para no programadores, muchos de los cuales pueden ser disuadidos de hacer uso de un herramienta de otra manera valiosa.

Independientemente de que los teóricos de la música decidan o no aumentar su invaluable 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[10] fue diseñado desde cero para compartir archivos de música entre aplicaciones y para 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 al los formatos de archivo utilizados por Finale y otros programas.

MusicXML se pretende un el 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)[11] es un intento de marcar objetos y eventos de música con un lenguaje basado en XML. La marcación de estos objetos debería permitir gestionar la música documentos para diversos fines, desde la teoría musical y la notación hasta rendimiento práctico. Este proyecto no está completo y está en progreso. El primer borrador de una posible DTD está disponible y se ofrecen algunos ejemplos de piezas de música marcadas con MML.

El enfoque es modular. Muchos módulos aún están incompletos y necesitan más investigación y atención.

Si una pieza musical está 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[12] 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 existentes y

[10] Good (2001)
[11] Steyn (2001)
[12] Clark y Tindale (2014)

potenciales tecnologías. Los usuarios interactúan con Flocking 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 generan nuevos instrumentos y puntajes 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.

2.3.5.1 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.

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".[13] Aunque Flocking está escrito en JavaScript, comparte con Lisp el enfoque expresar programas dentro de estructuras de datos que estén disponibles para su manipulación por otros programas.

Si bien el resumen expuesto no agota la lista de posibles referentes pertinentes a analizar y otros van aparecer mientras se vuelven relevantes, provee un criterio para proceder.

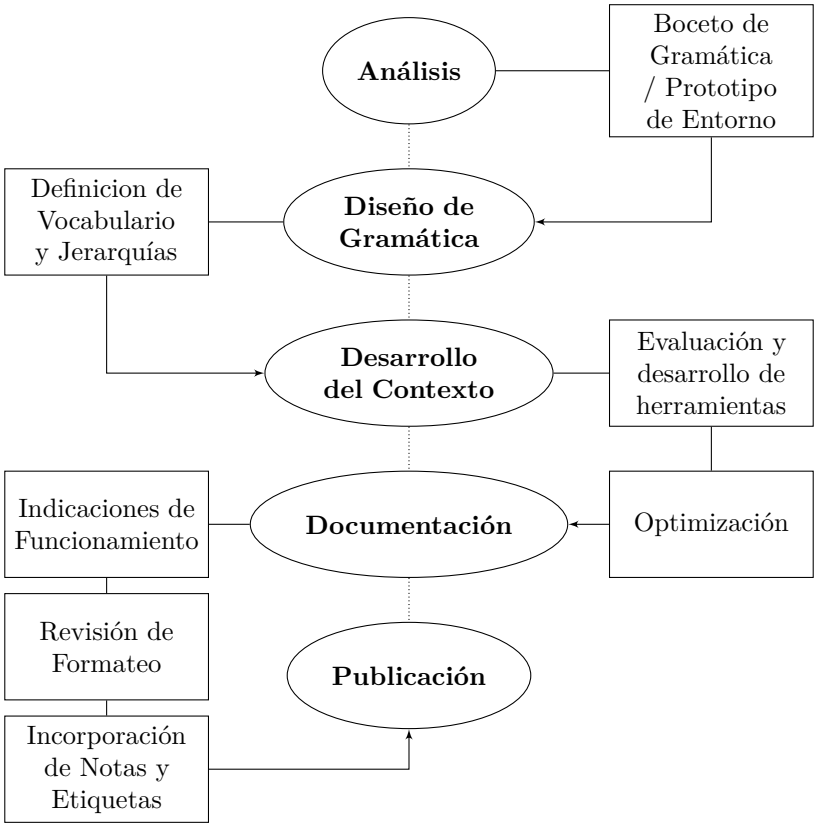
[13] Graham (2001)

3 Metodología

con lo previamente argumentando

En esta seccion se presentan los métodos optados para el desarrollo de la investigacion. Como recurso *didactico* se usa un gráfico para exponer el flujo de trabajo.

3.1 Diagrama de procedimiento



Sobre el desarrollo El entorno de producción musical que se pretende establecer estará principalmente integrando por:

3.2 Desarrollo

Sobre el desarrollo como conseguir el codigo. Instalación Uso

Sobre el desarrollo

3.2.1 YAML

El estándar YAML[14] como opción para serializar las definiciones de cada parte instrumental.

3.2.2 Python

La rutina de instrucciones principales será interpretada en el lenguaje Python[15] (en su ultima versión estable). Esta pieza de software estará basada en otros dos desarrollos: el módulo "*pyyaml*"[16] para analizar la información serializada, en combinación con la librería "*music21*"[17] que asistirá en las tareas de musicología. Ademas se incorporan algunos módulos de la "*Librería Estandar*",[18]

3.2.3 midiUTIL

midi

3.2.4 Otras herramientas

El editor de texto preferido para toda la actividad será VIM;[19] durante el desarrollo las versiones se controlarán con el sistema GIT[20] y el repositorio del proyecto se almacenará en un espacio online proveido por algún servicio del tipo GitLab.

[14] Varios (2018c)
[15] Rossum (2018)
[16] Varios (2018a)
[17] Cuthbert (2018)
[18] Varios (2018b)
[19] Moolenaar (2018)
[20] Torvalds (2018)

4 Resultados

4.1 Lenguaje formal

4.1.1 Estructura gramatical

referir a Metodología, YAML > La estructura principal la sintaxis gramatical de cada pista se basa en el formato de serialización de datos YAML[21] el cual delimta entre clave y valor con el cáracter ":" (dos puntos), mientras que la indentacion representa jerarquias, relacion de pertenecia entre parametros.

Multiples ficheros .yaml equivalen a multiples pistas en el resultado MIDI.

Describir Referencia y Recurrencia en YAML

«: *base (Para que otra pista herede estas propiedades)

4.1.2 Vocabulario

explicar q se va a describir cada palabra elegida para representar cada propiedad, etiqueta, el tipo de dato q es, un ejemplo y el valor defacto que se asigna

4.1.2.1 Propiedades de Pista

Los parametros generales de cada pista son tres: el rotulo, la paleta de unidades disponibles y el primer nivel de la forma musical. A partir del primer nivel estructural, las unidades se organizan entre ellas.

`nombre` (cadena de carcteres) default: `empty`

Titulo de la pista

```
nombre: 'Piano'
```

`complementos` (cadena de carcteres) default: `< empty >`

Ubicación de fichero con modulo de usuario.

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

`forma` (lista de cadenas de caracteres) default: `TO DO`

Macro estructura de la pista. Lista de unidades a ser sequenciadas. Corresponde a un elemento de la paleta.

```
forma: [
  'intro',
  'estrofa',
```

[21] Varios (2018c)

```
  'estribo',
  'coda',
]
```

`unidades` (diccionario) default: `TO DO`

Paleta de estructuras para secuenciar. En dos tipos de unidades, las que definen las estructuras minimas y las que invocan otras unidades ademas de sobrescribir o no alguno de sus parametros.

```yaml
unidades:
    base: &base
      clave:
      alteraciones: -2
      modo: 1
      registracion:  [
          -12,-10, -9, -7, -5, -3, -2,
            0,  2,  3,  5,  7,  9, 10,
           12, 14, 15, 17, 19, 21, 22,
           24
      ]
      alturas:  [ 1, 3, 5, 8 ]
      voces:
        - [ 8, 6 ]
        - [ 5 ]
        - [ 3 ]
      transportar: 60 # C
      transponer: 0
      duraciones: [ 1 ]
      bpm: 62
      metro: 4/4
      desplazar: 0
      reiterar: 0
      dinamicas: [ 1, .5, .4 ]
      canal: 3
      programa: 103
      controladores: [ 70:80, 70:90, 71:120 ]
    a: &a
      <<: *base
      metro: 2/4
      alturas: [ 1, 3,0, 5, 7, 8 ]
      duraciones: [ 1, .5, .5, 1, 1 ]
    b: &b
      <<: *base
      metro: 6/8
      duraciones: [ .5 ]
      alturas: [ 1, 2 ]
      voces: 0
```

```yaml
  revertir: [ 'duraciones', 'dinamicas' ]
  transponer: 3
  clave:
    alteraciones: 2
    modo: 1
  fluctuacion:
    min: .1
    max: .4
  desplazar: -1
b^:
  <<: *b
  dinamicas: [ .5, .1 ]
  revertir: [ 'alturas' ]
# Unidad de unidades ( UoUs )
# Propiedades sobrescriben a las de las unidades referidas
A:
  unidades: [ 'a', 'b' ]
  reiterar: 3
B: &B
  metro: 9/8
  unidades: [ 'a' , 'b^' ]
  desplazar: -0.75
B^:
  <<: *B
  voces: 0
  bmp: 89
  unidades: [ 'b', 'a' ]
  dinamicas: [ 1 ]
estrofa:
  unidades: [ 'A', 'B', 'B^' ]
coro:
  bpm: 100
  unidades: [ 'B', 'B^', 'a' ]
```

4.1.2.2 Propiedades de Unidad

Parametros por defecto para todas sas unidades, pueden ser sobrescritos.

`forma` (lista) default: `None`

Estructura de la Unidad

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

`clave` (diccionario) default: `alteraciones:0, modo: 0`

Armadura de clave Catidad de alteraciones en la armadura de clave y modo de la escala. Los numeros positivos representan sotenidos mientras que los se refiere a bemoles con números negativos. -2 = Bb, -1 = F, 0 = C, 1 = G, 2 = D, modo: 0 (mayor) Modo de la escala, 0 = Mayor o 1 = Menor (referir midi util doc, key signature)

```yaml
clave:
  alteraciones: -2
  modo: 1
```

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

Registración fija. Secuencia de intervalos a ser recorrida por el punteros de altura.

```yaml
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`

Transportar Ajuste de alturas. Semitonos

```yaml
transportar: 60 # C
```

`transponer` () default: `0`

Transponer puntero de registracion Ajuste de alturas pero dentro del set registracion.

```yaml
transponer: 1
```

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

Clave de compás Clave de metrica. representando una fracción (numerador / denominador).

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

`desplazar` (número decimal) default: `0`

Ajuste temporal Desfazage temporal del momento en el que originalmente comienza la unidad. offset : + / - offset con la ”posicion” original 0 es que

donde debe acontecer originalmente "-2" anticipar 2 pulsos o ".5" demorar medio pulso

```yaml
desplazar: -2
```

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

Repeticiones Catidad de veces q se toca esta unidad. Reiterarse a si misma, no es trasferible, no se hereda, caso contrario se reterarian los referidos.

```yaml
reiterar: 3
```

`dinamicas` (lista de números decimales) default: `[ 1 ]`

Dinámica Lista ordenada de dinámicas.

```yaml
dinamicas: [ 1, .5, .4 ]
```

`fluctuacion` (dicionario) default: `min:1, max:1`

Fluctuación fluctuciones dinámicas.

```yaml
fluctuacion:
  min: .3
  max: .7
```

`revertir` (lista de cadenas de caracteres) default: `[none]`

Sentido de las listas Revierte parametros del tipo lista. Deben corresponderse a la etiqueta de otro parametro del tipo lista.

```yaml
revertir: [ 'duraciones', 'dinamicas' ]
```

`canal` (número entero) default: `1`

Canal MIDI Número de Canal MIDI.

```yaml
canal: 3
```

4.1.2.3 Propiedades de Articulaciones

Parametros por defecto para todas sas unidades, pueden ser sobrescritos.

`alturas` (lista de números enteros) default: `[1]`

Punteros del set de registracion. Cada elemento equivale a el numero de intervalo.

```
alturas: [ 1, 3, 5, 8 ]
```

`voces` (lista de números enteros) default: `TO DO`

Superposicion de altura Apilamiento de alturas. Lista de listas, cada voz es un lista que modifica intervalo. voz + altura = numero de intervalo.

```
voces:
  - [ 8, 6 ]
  - [ 5 ]
  - [ 3 ]
```

`duraciones` (lista de números decimales) default: `[ 1 ]`

Duracion Lista ordenada de duraciones.

```
duraciones: [ 1, .5, .5, 1, 1 ]
```

`BPMs` (número entero) default: `60`

Pulso Tempo, Pulsos Por Minuto.

```
bpm: 62
```

`dinamicas` (lista de números decimales) default: `[ 1 ]`

Dinámica Lista ordenada de dinámicas.

```
dinamicas: [ 1, .5, .4 ]
```

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

Instrumento MIDI Número de Instrumento MIDI en el banco actual.

```
programa: 103
```

`controles` (lista de listas de pares) default: `None`

Cambios de control Secuencia de pares número controlador y valor a asignar.

```
controles:
  - [ 70 : 80,  71 : 90, 72 : 100 ]
  - [ 33 : 121, 51 : 120 ]
  - [ 10 : 80,  11 : 90, 12 : 100, 13 : 100 ]
```

4.2 Implementación

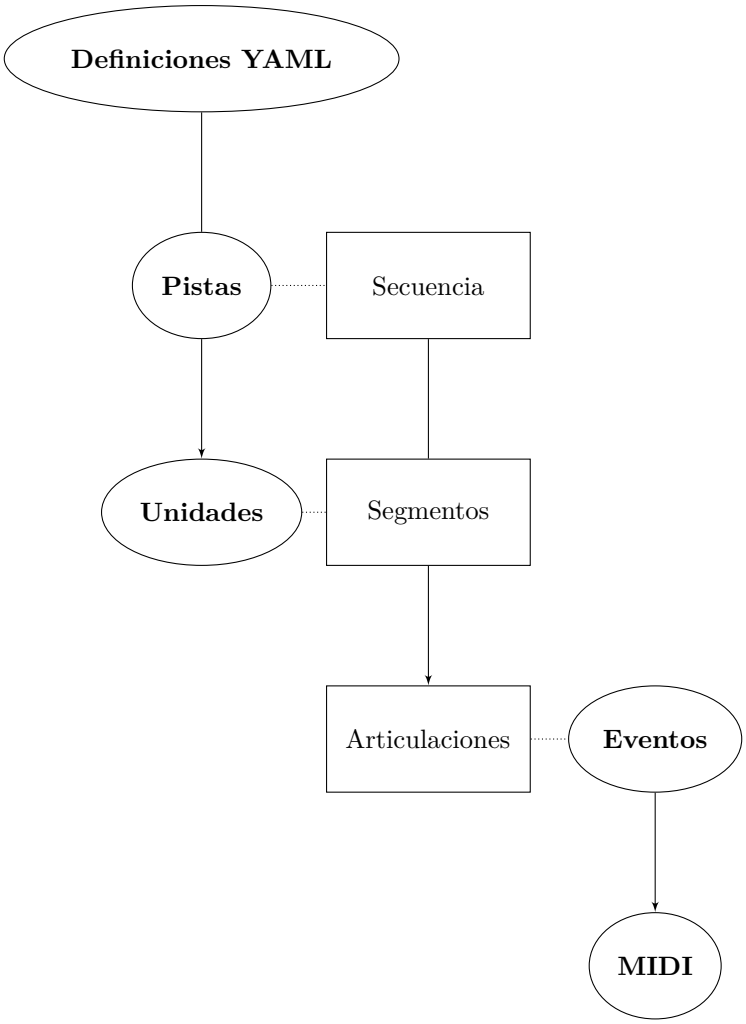
Introduccion a la subseccion

Aplicación y entorno de secuenciación

Lee archivos YAML como argumentos posicionales crea “pistas” a partir de ellos

esplicar estructura pista como flujo de eventos agrupados en segmentos agrupados en secciones

4.2.1 Diagrama de arquitectura



4.2.2 Secciones de pricipales del desarrollo

Explicacion de los bloques de codigo mas representativos

4.2.2.1 Modulo “Secuencia”

Loop principal que toma unidades previamente analizadas y llena lista de eventos.

4.2.2.2 Clase Pista

Clase Pista a partir de cada defefinicion de canal (.yml)

tienen un nombre parametros defaults de unidadad llamados “base” tiene una lista de unidades que se llama “macroforma” a partir de esta lista de busca en la paleta de unidades

a su vez cada unidad puede tener una lista de unidades a la que invoca arma un arbol de registros con las relaciones entre unidades arma una “sucecion” o “herencia” de parametros

repite la unidad (con sus hijas) segun parametro reiteracion agrega a los registros

Si la unidad actual tiene unidades sobrescribe los parametros de la unidad “hija” con los sucesion recursivamene busca hasta encontrar una sin unidades HIJAS Si la unidad altual NO tiene unidades finalmente mezcla el resultado con los defaults la secuencia hace secuencia de eventos

4.3 Demostraciones

Explicación de que ejemplo o demostración se va a discutir en cada sección.

4.3.1 Melodia Simple

Descripcion

4.3.1.1 YAML

Código

4.3.1.2 Partitura

Captura

4.3.1.3 Gráfico

ploteo

4.3.2 Multiples Canales

Descripcion

4.3.2.1 YAML

Códigos

4.3.2.2 Partitura

Capturas

4.3.2.3 Gráfico

ploteos

4.3.3 Polimetría

Paterns con duraciones no equivalentes

4.3.3.1 YAML

Códigos

4.3.3.2 Partitura

Capturas

4.3.3.3 Gráfico

ploteos

5 Concluciones

aplicaciones posibles en diferentes escenarios (online, archivología, livecodig) y varias disciplinas (IA, machine learning).

6 Apéndice

6.1 Secuencia

```
import os
from .pista import Pista
from .complementos import Complemento

class Secuencia:

  def __init__(
      self,
      defs,
      verbose = False,
      copyright = False
    ):
    self.defs = defs
    self.pistas = []
    self.verbose = verbose
    self.copyright = copyright

    for d in defs:
      pista = Pista(
        nombre    = d[ 'nombre' ],
        paleta    = d[ 'unidades' ],
        forma     = d[ 'forma' ],
        secuencia = self,
      )
      self.pistas.append( pista )

  @property
  def complementos( self ):
    complementos = []
    for d in self.defs:
      if 'complementos' in d:
       p = d[ 'complementos' ]
       if os.path.exists( p ):
         # TODO Tirar execpcion
         #and p not in Complemento.registro:
```

```
36          Complemento.registro.append( p )
37          c = Complemento( p )
38          complementos.append( c )
39      return complementos
40
41  @property
42  def eventos( self ):
43      """ A partir de cada definicion agrega una Pista. """
44      EVENTOS = []
45      for pista in self.pistas:
46
47          if self.verbose:
48              print( pista.verbose( self.verbose ) )
49
50          """ Generar track p/c pista """
51          delta = 0
52          track = pista.numero
53
54          """ Parametros de Pista Primer articulación de la parte, agregar
55          eventos fundamentales: pulso, armadura de clave, compás y programa.
56          """
57          EVENTOS.append([
58              'addTrackName',
59              track,
60              delta,
61              pista.nombre
62          ])
63          if self.copyright:
64              EVENTOS.append([
65                  'addCopyright',
66                  track,
67                  delta,
68                  self.copyright
69              ])
70
71          """ Loop principal:
72          Genera una secuencia de eventos MIDI lista de articulaciones.  """
73
74          for segmento in pista.segmentos:
75              canal = segmento.canal
76              #delta += segmento.desplazar
77
78              if delta < 0:
79                  raise ValueError( 'No se puede desplazar antes q el inicio' )
80                  pass
81
```

```
82          """ Agregar propiedades de segmento. """
83
84          if segmento.cambia( 'metro' ):
85            EVENTOS.append([
86              'addTimeSignature',
87              track,
88              delta,
89              segmento.metro[ 'numerador' ],
90              segmento.metro[ 'denominador' ],
91              segmento.metro[ 'relojes_por_tick' ],
92              segmento.metro[ 'notas_por_pulso' ]
93            ])
94
95          if segmento.cambia( 'bpm' ):
96            EVENTOS.append([
97              'addTempo',
98              track,
99              delta,
100             segmento.bpm,
101           ])
102
103         if segmento.cambia( 'clave' ):
104           EVENTOS.append([
105             'addKeySignature',
106             track,
107             delta,
108             segmento.clave[ 'alteraciones' ],
109             1, # multiplica por el n de alteraciones
110             segmento.clave[ 'modo' ]
111           ])
112
113         if segmento.afinacionNota:
114           EVENTOS.append([
115            'changeNoteTuning',
116             track,
117             segmento.afinacionNota[ 'afinaciones' ],
118             segmento.afinacionNota[ 'canalSysEx' ],
119             segmento.afinacionNota[ 'tiempoReal' ],
120             segmento.afinacionNota[ 'programa' ],
121           ])
122
123         if segmento.afinacionBanco:
124           EVENTOS.append([
125             'changeTuningBank',
126             track,
127             canal,
```

```
128             delta,
129             segmento.afinacionBanco[ 'banco' ],
130             segmento.afinacionBanco[ 'ordenar' ],
131           ])
132
133         if segmento.afinacionPrograma:
134           EVENTOS.append([
135             'changeTuningProgram',
136             track,
137             canal,
138             delta,
139             segmento.afinacionPrograma[ 'programa' ],
140             segmento.afinacionPrograma[ 'ordenar' ],
141           ])
142
143         if segmento.sysEx:
144           EVENTOS.append([
145            'addSysEx',
146             track,
147             delta,
148             segmento.sysEx[ 'fabricante' ],
149             segmento.sysEx[ 'playload' ],
150           ])
151
152         if segmento.uniSysEx:
153           EVENTOS.append([
154            'addUniversalSysEx',
155             track,
156             delta,
157             segmento.uniSysEx[ 'codigo' ],
158             segmento.uniSysEx[ 'subCodigo' ],
159             segmento.uniSysEx[ 'playload' ],
160             segmento.uniSysEx[ 'canal' ],
161             segmento.uniSysEx[ 'tiempoReal' ],
162           ])
163
164         if segmento.NRPN:
165           EVENTOS.append([
166            'makeNRPNCall',
167             track,
168             canal,
169             delta,
170             segmento.NRPN[ 'control_msb' ],
171             segmento.NRPN[ 'control_lsb' ],
172             segmento.NRPN[ 'data_msb' ],
173             segmento.NRPN[ 'data_lsb' ],
```

```
174             segmento.NRPN[ 'ordenar' ],
175           ])
176
177         if segmento.RPN:
178           EVENTOS.append([
179            'makeRPNCall',
180             track,
181             canal,
182             delta,
183             segmento.RPN[ 'control_msb' ],
184             segmento.RPN[ 'control_lsb' ],
185             segmento.RPN[ 'data_msb' ],
186             segmento.RPN[ 'data_lsb' ],
187             segmento.RPN[ 'ordenar' ],
188           ])
189
190         for articulacion in segmento.articulaciones:
191           """ Agrega cualquier cambio de parametro,
192           comparar cada uno con la articulacion previa. """
193
194           if articulacion.cambia( 'bpm' ):
195             EVENTOS.append([
196               'addTempo',
197               track,
198               delta,
199               articulacion.bpm,
200             ])
201
202           if articulacion.cambia( 'programa' ):
203             EVENTOS.append([
204                'addProgramChange',
205                track,
206                canal,
207                delta,
208                articulacion.programa
209             ])
210
211           if articulacion.letra:
212             EVENTOS.append([
213              'addText',
214               track,
215               delta,
216               articulacion.letra
217             ])
218
219           if articulacion.tono:
```

```
220             EVENTOS.append([
221                'addPitchWheelEvent',
222                track,
223                canal,
224                delta,
225                articulacion.tono
226             ])
227 
228           """ Agregar nota/s (altura, duracion, dinamica).
229           Si existe acorde en la articulación armar una lista con cada voz
230           superpuesta.  o una lista de solamente un elemento.  """
231           voces = [ articulacion.altura ]
232           if articulacion.acorde:
233             voces = articulacion.acorde
234 
235           for voz in voces:
236             EVENTOS.append([
237               'addNote',
238               track,
239               canal,
240               voz,
241               delta,
242               articulacion.duracion,
243               articulacion.dinamica
244             ])
245 
246           if articulacion.controles:
247             """ Agregar cambios de control """
248             for control in articulacion.controles:
249               for control, valor in control.items():
250                 EVENTOS.append([
251                  'addControllerEvent',
252                   track,
253                   canal,
254                   delta,
255                   control,
256                   valor,
257                 ])
258 
259           delta += articulacion.duracion
260     return EVENTOS
```

6.2 Pista

```
from argumentos import Excepcion
from .seccion  import Seccion
from .segmento import Segmento

class Pista:
  """
  Clase para cada definicion de a partir de archivos .yml
  PISTA > Secciones > Segmentos > Articulaciones
  """
  cantidad = 0

  def __str__( self ):
    o  = 'PISTA ' + str( self.numero) + ': '+ str( self.nombre  )
    return o

  def verbose( self, verbose = 0 ):
    if verbose > 0:
      o = str( self ) + ' '
      o = str( self ) + ' '
      o +=  '#' * ( 60 - len( o ))
      if verbose > 1:
        o += '\nELEM\t#\torden\tnivel\trecur\tnombre\n'
        for e in self.elementos:
          o += e.verbose( verbose )
          o += '\n'
      return o

  def __init__(
    self,
    nombre,
    paleta,
    forma,
    secuencia
  ):
    self.nombre     = nombre
    self.paleta     = paleta
    self.forma      = forma
    self.secuencia  = secuencia
    self.numero     = Pista.cantidad
    Pista.cantidad += 1

    self.secciones  = []
    self.segmentos  = []
```

```
44      self.seccionar( self.forma )
45
46    @property
47    def elementos( self ):
48      return sorted(
49        self.secciones + self.segmentos,
50        key = lambda x: x.numero
51      )
52
53    @property
54    def tiempo( self ):
55      # duracion en segundos
56      return sum( [ s.tiempo for s in self.segmentos ] )
57
58    """ Organiza unidades según relacion de referencia """
59    def seccionar(
60      self,
61      forma = None,
62      nivel = 0,
63      herencia = {},
64      referente = None,
65    ):
66      nivel += 1
67      """ Limpiar parametros q no se heredan.  """
68      herencia.pop( 'forma', None )
69      herencia.pop( 'reiterar', None )
70
71      for unidad in forma:
72        try:
73          if unidad not in self.paleta:
74            error = "PISTA: \"" + self.nombre + "\""
75            error += " NO ENCUENTRO: \"" + unidad + "\"  "
76            raise Excepcion( unidad, error )
77            pass
78          original = self.paleta[ unidad ]
79          sucesion = {
80            **original,
81            **herencia,
82          }
83          reiterar = 1
84          if 'reiterar' in original:
85            reiterar = original[ 'reiterar' ]
86          for r in range( reiterar ):
87            if 'forma' not in original:
88              segmento = Segmento(
89                pista       = self,
```

```
90              nombre      = unidad,
91              nivel       = nivel - 1,
92              orden       = len( self.segmentos ),
93              recurrencia = sum(
94                [ 1 for e in self.segmentos if e.nombre == unidad ]
95              ),
96              referente = referente,
97              propiedades = sucesion,
98            )
99            self.segmentos.append( segmento )
100         else:
101           seccion = Seccion(
102             pista       = self.nombre,
103             nombre      = unidad,
104             nivel       = nivel - 1,
105             orden       = len( self.secciones ),
106             recurrencia = sum(
107               [ 1 for e in self.secciones if e.nombre == unidad ]
108             ),
109             referente   = referente,
110           )
111           seccion.referidos = original['forma']
112           self.secciones.append( seccion )
113           elemento = seccion
114           self.seccionar(
115             original[ 'forma' ],
116             nivel,
117             sucesion,
118             seccion,
119           )
120     except Excepcion as e:
121         print( e )
```

6.3 Elemento

```
class Elemento():
  """
  Pista > ELEMENTOS
  Metaclase base para, Secciones, Segmentos
  """
  cantidad = 0
  def __str__( self ):
    o  = str( self.numero ) + '\t'
    o += str( self.orden ) + '\t'
    o += str( self.nivel )  + '\t'
    o += str( self.recurrencia ) + '\t'
    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 = Elemento.cantidad
    Elemento.cantidad += 1
```

6.4 Sección

```
from .elemento import Elemento

class Seccion( Elemento ):
  cantidad = 0
  """ Pista > SECCION > Segmentos > Articulaciones """

  def verbose( self, verbose = 0 ):
    o = self.tipo + ' '
    o += str( self.numero_seccion ) + '\t'
    o += str( self ) + ' '
    o += '=' * ( 18 - ( len( self.nombre ) + self.nivel ) )
    return o

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

6.5 Segmento

```
import math
from .elemento import Elemento
from .articulacion import Articulacion
from .complementos import Complemento


class Segmento( Elemento ):
  """
  Secuencia > Pista > Secciones > SEGMENTOS > Articulaciones
  Conjunto de Articulaciones
  """
  cantidad = 0

  defactos = {

    # Propiedades de Segmento
    'canal'    : 0,
    'revertir' : None,
    'NRPN'     : None,
    'RPN'      : None,

    # Props. que NO refieren a Canal especifico
    # ¿a Meta track?  Igualmente midiutil las manda a canal 16...
    'metro'             : '4/4',
    'alteraciones'      : 0,
    'modo'              : 0,
    'afinacionNota'     : None,
    'afinacionBanco'    : None,
    'afinacionPrograma' : None,
    'sysEx'             : None,
    'uniSysEx'          : None,

    # Procesos de Segmento
    'transportar' : 0,
    'transponer'  : 0,
    'reiterar'    : 1,

    # Propiedades de Articulacion
    'BPMs'         : [ 60 ],
    'programas'    : [ None ],
    'duraciones'   : [ 1 ],
    'dinamicas'    : [ 1 ],
    'registracion' : [ 1 ],
```

```
    'alturas'       : [ 1 ],
    'letras'        : [ None ],
    'tonos'         : [ 0 ],
    'voces'         : None,
    'controles'     : None,

  }

  def verbose( self, verbose = 0 ):
    o = self.tipo + ' '
    o += str( self.numero_segmento) + '\t'
    o += str( self ) + ' '
    o += '-' * ( 18 - (len( self.nombre ) + self.nivel))
    if verbose > 2:
      o += '\nARTICULACIONES\n'
      o += '#\tord\tbpm\tdur\tdin\talt\tltr\tton\tctrs\n'
      for a in self.articulaciones:
        o += str( a )
    return o

  def __init__(
    self,
    pista,
    nombre,
    nivel,
    orden,
    recurrencia,
    referente,
    propiedades
  ):
    Elemento.__init__(
      self,
      pista,
      nombre,
      nivel,
      orden,
      recurrencia,
      referente
    )
    self.numero_segmento = Segmento.cantidad
    Segmento.cantidad += 1
    self.tipo = 'SGMT'
    self.props = {
        **Segmento.defactos,
        **propiedades
    }
```

```
90      """ PRE PROCESO DE SEGMENTO """
91
92      """ Cambia el sentido de los parametros de
93      articulacion """
94      self.revertir = self.props[ 'revertir' ]
95      if self.revertir:
96        if isinstance( self.revertir , list ):
97          for r in self.revertir:
98            if r in self.props:
99              self.props[ r ].reverse()
100       elif isinstance( self.revertir , str ):
101         if revertir in self.props:
102           self.props[ self.revertir ].reverse()
103
104     self.canal             = self.props[ 'canal' ]
105     self.reiterar          = self.props[ 'reiterar' ]
106     self.transponer        = self.props[ 'transponer' ]
107     self.transportar       = self.props[ 'transportar' ]
108     self.alteraciones      = self.props[ 'alteraciones' ]
109     self.modo              = self.props[ 'modo' ]
110     self.afinacionNota     = self.props[ 'afinacionNota' ]
111     self.afinacionBanco    = self.props[ 'afinacionBanco' ]
112     self.afinacionPrograma = self.props[ 'afinacionPrograma' ]
113     self.sysEx             = self.props[ 'sysEx' ]
114     self.uniSysEx          = self.props[ 'uniSysEx' ]
115     self.NRPN              = self.props[ 'NRPN' ]
116     self.RPN               = self.props[ 'RPN' ]
117     self.registracion      = self.props[ 'registracion' ]
118     self.programas         = self.props[ 'programas' ]
119     self.duraciones        = self.props[ 'duraciones' ]
120     self.BPMs              = self.props[ 'BPMs' ]
121     self.dinamicas         = self.props[ 'dinamicas' ]
122     self.alturas           = self.props[ 'alturas' ]
123     self.letras            = self.props[ 'letras' ]
124     self.tonos             = self.props[ 'tonos' ]
125     self.voces             = self.props[ 'voces' ]
126     self.capas             = self.props[ 'controles' ]
127
128     self.bpm = self.BPMs[0]
129     self.programa = self.programas[0]
130
131
132     """ COMPLEMENTOS
133         Pasar propiedades por metodos de usuario
134     """
135     for complemento in self.pista.secuencia.complementos:
```

```
136         for metodo in dir( complemento.modulo ):
137           if metodo in self.props:
138             for clave in self.props[ metodo ]:
139               original = getattr( self, clave )
140               argumentos = self.props[ metodo ][ clave ]
141               #print( metodo, ':', clave, argumentos )
142               modificado = getattr(
143                  complemento.modulo,
144                  metodo,
145               )( original, argumentos )
146               setattr( self, clave, modificado )
147
148
149   @property
150   def precedente( self ):
151     n = self.orden
152     o = self.pista.segmentos[ n - 1 ]
153     return o
154
155   def obtener( self, key ):
156       try:
157         o = getattr( self, key )
158         return o
159       except AttributeError as e:
160         return e
161
162   def cambia(
163       self,
164       key
165     ):
166     este = self.obtener( key )
167     anterior = self.precedente.obtener( key )
168     if (
169       self.orden == 0
170       and este
171     ):
172       return True
173     return anterior != este
174
175   @property
176   def tiempo( self ):
177     # duracion en segundos
178     return sum( [ a.tiempo for a in self.articulaciones ] )
179
180   @property
181   def metro( self ):
```

```
    metro = self.props[ 'metro' ].split( '/' )
    denominador = int(
      math.log10( int( metro[ 1 ] ) ) / math.log10( 2 )
    )
    return {
      'numerador'         : int( metro[ 0 ] ),
      'denominador'       : denominador,
      'relojes_por_tick' : 12 * denominador,
      'notas_por_pulso'   : 8,
    }

  @property
  def clave( self ):
    return {
      'alteraciones' : self.alteraciones,
      'modo' : self.modo
    }

  @property
  def ganador( self ):
    """ Evaluar que propiedad lista es el que mas valores tiene.  """
    self.ganador_voces = [ 0 ]
    if self.voces:
      self.ganador_voces = max( self.voces, key = len)
    self.ganador_capas = [ 0 ]
    if self.capas:
      self.ganador_capas = max( self.capas , key = len)

    candidatos = [
      self.dinamicas,
      self.duraciones,
      self.alturas,
      self.letras,
      self.tonos,
      self.BPMs,
      self.programas,
      self.ganador_voces,
      self.ganador_capas,
    ]
    return  max( candidatos, key = len )

  @property
  def cantidad_pasos( self ):
    return len( self.ganador )

  @property
```

```
228   def articulaciones( self ):
229
230     """ Consolidar "articulacion"
231     combinar parametros: altura, duracion, dinamica, etc. """
232     o = []
233     for paso in range( self.cantidad_pasos ):
234       """ Alturas, voz y superposición voces. """
235       altura = self.alturas[ paso % len( self.alturas ) ]
236       acorde = []
237       nota   = altura
238       """ Relacion: altura > puntero en el set de registracion;
239       Trasponer dentro del set de registracion, luego Transportar,
240       sumar a la nota resultante. """
241       n = self.registracion[
242         ( ( altura - 1 ) + self.transponer ) % len( self.registracion )
243       ]
244       nota = self.transportar + n
245       """ Armar superposicion de voces. """
246       if self.voces:
247         for v in self.voces:
248           voz = (
249             altura + ( v[ paso % len( v ) ] ) - 1
250           ) + self.transponer
251           acorde += [
252             self.transportar +
253             self.registracion[ voz % len( self.registracion ) ]
254           ]
255       """ Cambios de control. """
256       controles = []
257       if self.capas:
258         for capa in self.capas:
259           controles += [ capa[ paso % len( capa ) ] ]
260       """ Articulación a secuenciar. """
261       articulacion = Articulacion(
262          segmento  = self,
263          orden     = paso,
264          bpm       = self.BPMs[ paso % len( self.BPMs ) ],
265          programa  = self.programas[ paso % len( self.programas ) ],
266          duracion  = self.duraciones[ paso % len( self.duraciones ) ],
267          dinamica  = self.dinamicas[ paso % len( self.dinamicas ) ],
268          nota      = nota,
269          acorde    = acorde,
270          tono      = self.tonos[ paso % len( self.tonos ) ],
271          letra     = self.letras[ paso % len( self.letras ) ],
272          controles = controles,
273       )
```

```
274         o.append( articulacion )
275     return o
276
```

6.6 Articulación

```
class Articulacion:

  """ Pista > Segmentos > ARTICULACIONES """

  cantidad = 0

  def __str__( self ):
    o  = str( self.numero ) + '\t'
    o += str( self.orden ) + '\t'
    o += str( self.bpm ) + '\t'
    o += str( self.duracion ) + '\t'
    o += str( self.dinamica ) + '\t'
    o += str( self.altura ) + '\t'
    o += str( self.letra ) + '\t'
    o += str( self.tono) + '\t'
    o += str( self.controles) + '\n'
    return o

  def __init__(
    self,
    segmento,
    orden,
    bpm,
    programa,
    duracion,
    dinamica,
    nota,
    acorde,
    tono,
    letra,
    controles,
  ):
    self.numero = Articulacion.cantidad
    Articulacion.cantidad += 1

    self.segmento  = segmento
    self.orden     = orden
    self.bpm       = bpm
    self.programa  = programa
    self.tono      = tono
    self._dinamica  = dinamica
    self.duracion  = duracion
    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
84    @property
85    def dinamica( self ):
86      viejo_valor = self._dinamica
87      viejo_min = 0
88      viejo_max = 1
89      nuevo_min = 0
```

```
90      nuevo_max = 126
91      nuevo_valor = (
92        ( viejo_valor - viejo_min ) / ( viejo_max - viejo_min )
93      ) * ( nuevo_max - nuevo_min) + nuevo_min
94      return int( min( max( nuevo_valor, nuevo_min ), nuevo_max ) )
```

6.7 Complementos

```
import importlib.util as importar

class Complemento:

    """ Interfaz minimalista para complementos de usuario.

        Declarar ubicación del paquete en propiedades de track.
        complementos: 'enchufes.py'

        En propiedades de segmento invocar metodo y subscribir, propiedad y
        argumentos.

        metodo: # en: enchufes.py
          propiedad_a_manipular1: argumentos
          propiedad_a_manipular2: argumentos
        fluctuar:
          dinamicas: .5
          alturas: 2
    """

    cantidad = 0
    registro = []

    def __str__(
      self,
    ):
      return self.nombre

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

7 Bibliografía

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 DE ROSE, S.J., 1987. Markup Systems and the Future of Scholarly Text Processing. *Communications of the ACM* [en línea], vol. 30, no. 11, pp. 933-47. DOI 10.1145/32206.32209. Disponible en: http://www.xml.coverpagess.org/coombs.html.

CUTHBERT, M.S., 2018. music21: a toolkit for computer-aided musicology. [en línea]. Disponible en: http://web.mit.edu/music21.

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.

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

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.

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

MOOLENAAR, B., 2018. VIM. [en línea]. Disponible en: https://www.vim.org/docs.php.

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., 2018. GIT. [en línea]. Disponible en: https://git-scm.com/docs.

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

VARIOS, A., 2018b. The Pyhton Standar Library. [en línea]. Disponible en: https://docs.python.org/3/library/index.html.

VARIOS, A., 2018c. YAML Ain't Markup Language. [en línea]. Disponible en: http://yaml.org/.

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.

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.