Universidad Nacional de Quilmes

Escuela de Artes

LICENCIATURA EN MÚSICA Y TECNOLOGÍA

Director de Carrera: Esteban Calcagno

Tesis de Grado

Presentada por: Lisandro Fernández

Director de Tesis: no sabemos todavia

Abstract

Definición de estructura y gramática de plan de obra musical, basada en texto serializado legible y desarrollo de un sistema de herramientas CLI para generar sequencias MIDI a partir de las mismas.

Septiembre 2017

Buenos Aires, Argentina

Contents

1. Resumen

El presente plan propone la definición de una gramática neutral para representar un discurso musical en estructura de hojas de análisis o planes de obra, basada en texto plano[1][2] serializado y legible[3], acompañada por el desarrollo de un sistema de herramientas para interfaz de línea de comandos (Command Line Interface)[4] que genere secuencias musicales en el estándar MIDI manipulando la información representada en estos ficheros

Se documentará[5] este desarrollo para su correcta publicación enmarcada bajo premisas del software libre[6][7].

2. Justificacion

2.1 ¿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 & Thomas, 1999)

- Leverage

[1] Hunt & Thomas (1999)
[2] Leek (2017)
[3] Coombs, Renear, & DeRose (1987)
[4] Hunt & Thomas (1999)
[5] Kernighan & Plaguer (1978)
[6] Varios (2001)
[7] Yzaguirre (2016)

- **Soportado en multiples plataformas.** Cada sistema operativo tiene al menos un editor de texto y son todos compatibles hasta la codificación del texto.
- **Fácil de manipular.** Procesar cadenas de caracteres es uno de los procesos mas elementales que se pueden realizar en un sistema informático.
- **Fácil de mantener.** Comparado con alternativas mas complejas, el texto plano es lo mas sencillo ante la necesidad de actualizar información o de realizar cualquier tipo de cambio o ajuste.
- Easier testing
- **Liviano.** Aspecto clave cuando los recursos a nivel de sistema son limitados, como por ejemplo poca capacidad de procesamiento/almacenamiento, conexiones lentas o sin acceso a una interfaz gráfica.
- **Compatible con el avance.**
- Insurance against obsolescence Independientemente de la próxima plataforma, si tenemos nuestros documentos almacenados en texto plano, serán fáciles de interpretar.

2.2 ¿Por qué la Interfaz de Linea de Comandos?

- Esta en todos los sistemas
- Primer estado utilizable del sistema
- Agilidad
- Misma interfaz para múltiples aplicaciones
- Resultados reproducibles
- Pipeline
- Tradición
- Menos recursos
- Acceso remoto
- VIM

3. Objetivos

Las motivaciones principales del desarrollo que este plan propone son:

- Elaborar una herramienta de composición musical que prescinda de interfaz gráfica.

- Unificar el proceso de planificación de obra musical con el de secuenciación MIDI.
- Exponer ventajas de la operación de sistemas a través de la Interfaz de Linea de Comandos.
- Acercar a músicos a la composición musical asistida por ordenadores.

4. Marco referencial

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.

4.1 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.

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

[8] Selfridge-Field (1997)

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.

4.1.2 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.

4.2 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++.

4.2.1 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)

4.2.2 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, que el usuario debe aprender a utilizar en combinación para explotar plenamente el potencial del sistema.

4.3 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.

4.4 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)

[10] Good (2001)
[11] Steyn (2001)

- 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.

4.5 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 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.

4.5.1 Como funciona Flocking

El núcleo del framework Flocking consiste en varios componentes interconectados que proporcionan la capacidad esencial de interpretar e instanciar generadores de unidades, producir flujos de muestras y programar procesos. Los principales componentes de Flocking incluyen:

1. el intérprete Flocking, que analiza e instancia sintetizadores, generadores de unidad y búferes
2. el medio ambiente, que representa el audio general y su configuración
3. Audio Strategies, que son las salidas de audio conectables (vinculados a los backends como la API de audio web o ALSA en Node.js)
4. Generadores unitarios (ugens), que son funciones primitivas generadoras de las muestras utilizadas para producir sonido
5. Sintetizadores, que representan instrumentos y colecciones en la lógica de generación de señales

[12] Clark & Tindale (2014)

6. el Programador, que gestiona el cambio secunecial (basado en el tiempo) eventos en un sintetizador

4.5.2 Programación declarativa

Arriba, describimos Flocking como un marco **declarativo**. Esta característica es esencial para comprender su diseño.

La programación declarativa se puede entender en el contexto de Flocking como 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, que en la secuenciación de bajo nivel y el flujo de control. Tradicional los estilos de programación imperativos suelen una"audiencia de uno" -el compilador. Aunque el código es a menudo compartido entre varios desarrolladores, no suele ser comprendidos o manipulados por programas distintos de los compilador.

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. los

La familia de lenguas Lisp es un ejemplo bien conocido de este enfoque. Paul Graham describe la naturaleza declarativa de Lisp, diciendo que "no tiene sintaxis. Escribes programas en los árboles de análisis... [que] son totalmente accesibles a sus programas. Puede escribir programas que los manipulen... programas que escriben programas". Aunque Flocking está escrito en JavaScript normal, comparte con Lisp el enfoque expresar programas dentro de estructuras de datos que estén disponible para su manipulación por otros programas.

5. Metodología

- Análisis de caso
  - Boceto de sintaxis
  - Prototipo de aplicación
  - Encuestas y consultas a músicos compositores y teóricos
- Diseño de Gramática
  - Sintaxis basada en YAML
  - Terminología de análisis/planificación musical

- Desarrollo
  - Perl
  - GIT
- Documentación
  - Modulo POD
  - Paginas MAN, PDF, HTML, etc.
- Release en repositorios públicos

6. Cronograma de Trabajo

	Tiempo Estipulado	Fechas Tentativas
Boceto de sintaxis	1 semana	Del XX al XX de XXXXXX
Prototipo aplicación	2 semanas	Del XX al XX de XXXXXX
Entrevistas y consultas	1 semana	Del XX al XX de XXXXXX
Definición de gramática	5 semanas	Del XX al XX de XXXXXX
Dasarrollo de herramientas	7 semanas	Del XX al XX de XXXXXX
Pruebas y optimización	3 semanss	Del XX al XX de XXXXXX
Documentación	5 semanas	Del XX al XX de XXXXXX

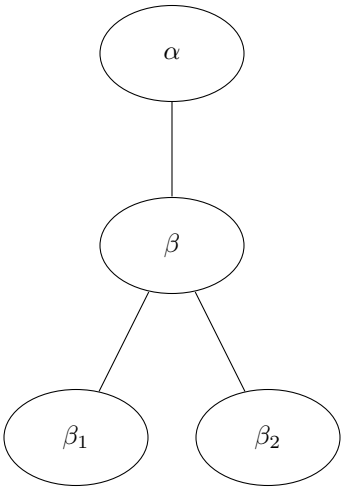

Figure 1: example graph.

7. Bibliografía

Clark, C., & Tindale, A. (2014). Flocking: A Framework for Declarative Music-Making on the Web. *The Joint Proceedings of the ICMC and SMC*, *1*(1), 1150–57.

Coombs, J. H., Renear, A. H., & DeRose, S. J. (1987). Markup Systems and the Future of Scholarly Text Processing. *Communications of the ACM*, *30*(11), 933–47. https://doi.org/10.1145/32206.32209

Good, M. (2001). MusicXML: An Internet-Friendly Format for Sheet Music. *Proceedings of XML*. Retrieved from http://michaelgood.info/publications/music/musicxml-an-internet-friendly-format-for-sheet-music/

Hunt, A., & Thomas, D. (1999). *The Pragmatic Programmer: From Journeyman to Master*. (, &, Eds.) (pp. 72–99). The Pragmatic Bookshelf.

Kernighan, B. W., & Plaguer, P. J. (1978). *The Elemenets Of Programing Style* (pp. 141–53). McGRAW-HILL BOOK COMPANY.

Leek, J. (2017, June 13). The future of education is plain text. Retrieved from https://simplystatistics.org/2017/06/13/the-future-of-education-is-plain-text

Selfridge-Field, E. (Ed.). (1997). *Beyond MIDI: The Jandbok of Musical Codes* (pp. 402–445). The MIT Press.

Steyn, J. (2001, December 21). Music Markup Language. Retrieved from https://steyn.pro/mml

Varios, A. (2001, December 21). ¿Que es el Software Libre? Retrieved from https://www.gnu.org/philosophy/free-sw.es.html

Wild, J. (1996). A Review of the Humdrum Toolkit: UNIX Tools for Musical Research, created by David Huron. *Music Theory Online*, *2*(7).

Yzaguirre, G. (2016, November 20). Manifiesto del Laboratorio de Software Libre. Retrieved from https://labsl.multimediales.com.ar/Manifiesto_del_Laboratorio_de_Software_Libre_.html