Responsable de proyecto:

Representado por:

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.

Universidad Nacional de Quilmes
Taller de Tesis

Docente: **Esteban Calcagno**

Alumno: **Lisandro Fernández**

Introducción

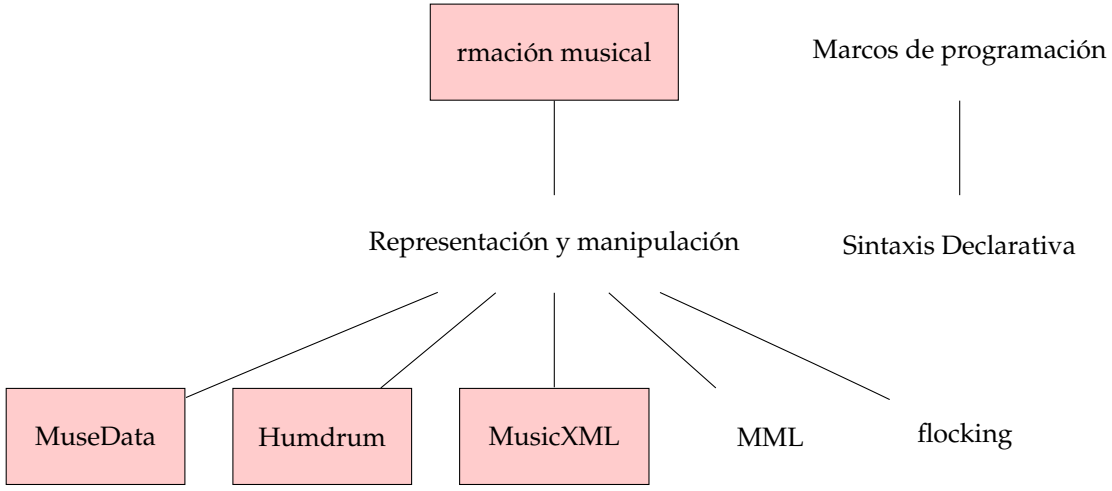

Figure 1:

1. MuseData

- Universidad de Stanford
  - Centro de Investigación Asistida por Computador en Humanidades (CCARH).
    - Creado por Walter Hewlett.
- Base de datos MuseData[1]
  - Estructura y a la vez el sistema de codificación
- Archivos => múltiples formatos.
- El archivo MuseData está diseñado para soportar aplicaciones de sonido, gráficos y análisis.

[1] Selfridge-Field (1997)

1.1 Organización de archivos MuseData

Los archivos MuseData

- basados en ASCII => cualquier editor de texto.
- Organizados en base a las partes.
- El número de archivos y el formato puede variar por movimiento y por trabajo
- Archivos para los diferentes movimientos de una composición se encuentran en directorios separados
- Un movimiento de una composición es típicamente encontrado
  - Rótulo: número de movimiento, p. 01, 02, etc.
  - Dividido en varios archivos, agrupados en ese directorio para ese movimiento.
  - Las partes de los archivos
    - Etiqueta 01 para la primera parte, 02 para la 2da parte de la partitura, etc.

1.2 La representación MuseData de información musical

- Propósito de la sintaxis:
  - Representar el contenido lógico de una pieza musical de una modo neutral.
- Actualmente bases de datos de texto varios compositores
  - J.S. Bach, Beethoven, Corelli, Handel, Haydn, Mozart, Telemann y Vivaldi
- Se utilicen para la
  - Impresión de música
  - Análisis musical
  - Producción de archivos de sonido digitales.
- Destinado para a ser genérico
  - Diseño para representar tanto información de
    - Notación
    - Sonido
- Se han desarrollado piezas de software de diversos tipos con el fin de probar su eficacia
  - Aplicaciones imprimir resultados
  - Partes para ser utilizadas por editores profesionales de música
  - Así como también compilar archivos MIDI
  - Facilitar las búsquedas rápidas de los datos de patrones específico
    - Rítmicos
    - Melódicos

        * Armónicos

- No se pretende que la representación esté completa
    - Registros MuseData servirían como archivos de origen para generar tanto
        * Documentos gráficos (específicamente de página)
        * Archivos de performance MIDI,
    - 2 razones para esta postura:
        * no es la partitura sino el contenido lógico de la partitura lo que codifica.
            · Codificar la puntuación significaría codificar la posición exacta contendría más información que la que el compositor pretende transmitir.
        * No se puede anticipar todos los usos a q podrían darse Si podemos asegurar que c/suario tendrá sus propias necesidades especiales y preferencias.

1.3.1 Ejemplo

```
04/16/93 E. Correia
WK#:581       MV#:3c
Breitkopf & H\3artel, Vol. 13
Clarinet Quintet
Trio II
Violoncello
1 0
Group memberships: sound, score
sound: part 5 of 5
score: part 5 of 5
$  K:3   Q:2   T:3/4   C:22
rest   2        q
measure 1
A3     2        q     d        p
rest   2        q
rest   2        q
measure 10
rest   6
measure 11
E2     2        q     u        (.
E2     2        q     u         .
E2     2        q     u        ).
measure 12
A2     2        q     u
rest   2        q
mheavy4         :||:
/END
```

2. Humdrum

David Huron creó Humdrum[2] 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]Wild (1996)

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.

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.

3. MusicXML

MusicXML[3] 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.

[3]Good (2001)

4. Music Markup Language

El Lenguaje de Marcado de Música (MML)[4] 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.

[4] Steyn (2001)

5. Flocking

Flocking[5] 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.

[5]Clark y Tindale (2014)

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 *Ecosistema*, 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. *Unit Generators* (ugens), que son funciones primitivas generadoras de las muestras utilizadas para producir sonido
5. *Synths* (sintetizadores) que representan instrumentos y colecciones en la lógica de generación de señales
6. el *Scheduler* (programador ó secuenciador), que gestiona el cambio secunecial (basado en el tiempo) eventos en un sintetizador

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 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". 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.

5.2.1 Ejemplo

```
// Create a new synth consisting of a sine wave,
// modulating its amplitude slowly with another sine wave.
var synth = flock.synth({
    synthDef: {
        id: carrier,
        ugen: flock.ugen.sinOsc,
        freq: 440,
        mul: {
            id: mod,
            ugen: flock.ugen.sinOsc,
            freq: 1.0,
            mul: 0.25
        }
    }
});
```

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. 1150-57.

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/.

SELFRIDGE-FIELD, E., 1997. *Beyond MIDI: The Jandbok of Musical Codes*. S.l.: The MIT Press. ISBN 9780262193948.

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

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.