Escuela Politécnica Superior

Asistente de composición de música dodecafónica para OpenMusic

Grado en Ingeniería Informática

Trabajo Fin de Grado

Autor:
Leopoldo Pla Sempere

Tutor/es:
Carlos Pérez Sancho

Universitat d'Alacant
Universidad de Alicante

Junio 2014

Asistente de composición de música dodecafónica para OpenMusic

Leopoldo Pla Sempere (lps34 at alu dot ua dot es)
Universidad de Alicante

13 de junio de 2014

Justificación y objetivos

La principal razón por la que he realizado este trabajo ha sido por aprovechar los conocimientos de los dos estudios realizados, el Grado Profesional de música y el Grado en Ingeniería Informática en un proyecto que pueda ser útil para ambas áreas, ayudando a los músicos a componer y sentando una base para los ingenieros que quieran hacer un asistente de composición o que tengan ideas para mejorarlo. También una de las razones por la cuál he elegido esta área de la música como es la composición, y más concretamente el dodecafonismo, es la curiosidad sobre la programación en entornos de composición musical, la mejora e integración de los ordenadores en tareas mecánicas o programables como la composición de música dodecafónica y la relación de la música dodecafónica con el álgebra lineal.

El objetivo principal que se quiere alcanzar con este trabajo es, como he mencionado antes, mejorar el proceso de composición de música dodecafónica proporcionando al compositor ideas y fragmentos generados con las restricciones que él desee así como retroalimentar al programa proponiendo más restricciones o formas compositivas que pueda crear para que otros ingenieros puedan mejorar este sistema.

Agradecimientos

Agradecer este trabajo a mi familia por la ayuda y el apoyo que me han dado para llegar a crear este documento que mezcla dos mundos muchas veces duros de conllevar.

A mi tutor, Carlos, por orientar de una manera sencilla el proyecto y ser honesto con los objetivos propuestos.

A mis compañeros y profesores del grado, con los que he compartido muy buenos momentos y mucha experiencia profesional.

Dedicado a
Laura

Resumen

El asistente de composición de música dodecafónica es un conjunto de capas software implementadas sobre varias tecnologías, entre ellas OpenMusic y Python, para crear obras dodecafónicas según unas restricciones impuestas por el usuario. El sistema es capaz de obtener una serie dodecafónica en base a una semilla propuesta cumpliendo además ciertas restricciones para luego obtener todas las series derivadas de ella y componer con estas series y otras restricciones.

Se genera una partitura con ritmos, saltos de octava y silencios además de poder exportar el resultado a formato MusicXML pudiéndose abrir en cualquier editor de partituras actual. Aquí presentaremos y mostraremos en imágenes el asistente con algunos ejemplos así como de exponer diferentes partes de su desarrollo. También se mostrará el diseño y la programación del programa bajo las diferentes tecnologías utilizadas y se sugieren enfoques para el futuro desarrollo.

Índice general

Índice de figuras

Índice de cuadros

Capítulo 1

Introduction

Con el avance del procesamiento de la información, han surgido herramientas para músicos que ayudan a realizar tareas tanto a nivel de tratamiento del sonido, edición y síntesis como a nivel de aprendizaje musical y rítmico, abarcando tanto a músicos aficionados o profesionales como a ingenieros de diversas ramas. Una de las vías que mantiene unidos a los músicos profesionales y los ingenieros es la música por ordenador (o la representación musical), donde los grupos que tratan esta materia investigan y desarrollan estructuras, lenguajes y paradigmas adaptados a la música.

Las áreas que más se tratan en estos grupos actualmente son la clasificación de música, análisis y composición automática (siendo esta la que más tiempo se lleva investigando y desarrollando). Todas estas áreas de la música conllevan el conocimiento de técnicas como el reconocimiento de patrones (*pattern recognition*) o el aprendizaje máquina (*machine learning*).

Capítulo 2

Marco teórico

2.1. Dodecafonismo

Arnold Schoenberg (1874-1951) fue un compositor, teórico musical y pinto austriaco de origen judío. En 1905 creó un método de composición basado en la ausencia de centro tonal, el atonalismo, ya que para Arnold la tonalidad era un recurso que se había explotado al máximo y se debía buscar alternativas armónicas para la composición. Este método aumentó la dificultad de composición de obras extensas ya que era muy limitado para crear estructuras coherentes de larga duración sin que estuviesen apoyadas en un texto que las cohesionase. Este movivo provocó que entre 1914 y 1921 no publicara ningún escrito hasta que creó una solución. Entre 1921 y 1923 inició un método de composición musical derivado del atonalismo que solventaba sus limitaciones: el dodecafonismo (*twelve-tone method*)[3]. La primera obra que publicó con este método es la Suite para piano Op. 25.

Schoenberg creó este método por el mismo hecho histórico por el que se creó la música modal o la música tonal: por el desarrollo natural e histórico de la música.

> *Este compromiso que se llama 'sistema temperado', representa una tregua por un tiempo indeterminado. Pues esta reducción de las relaciones naturales a las manejables no podrá resistir indefinidamente la evolución musical; y cuando el oído lo exija se enfrentará con el problema. Y entonces nuestra escala quedará asumida en una ordenación superior, como lo fueron los modos eclesiásticos en los modos mayor y menor. Y no podemos prever si habrá cuartos, octavos, tercios de tono o (como piensa Busoni) sextos de tono, o si iremos directamente a una escala de 53 sonidos como la establecida por el Dr. Robert Neuman.[4]*

Este método de composición es un tipo de serialismo (composición basada en series de notas, de ritmos, de dinámicas...) en el que no se busca la importancia de unas notas sobre otras (que es la base de la tonalidad). Es decir, busca lo contrario que la música tonal (por ejemplo, la música clásica, o la música rock), que tiene normalmente un tono predominante y unas notas dominantes sobre otras (la dominante sobre la tónica, que forman la cadencia perfecta que define el tono y crea centros tonales). Para evitar este énfasis, se crea una serie con las doce notas de la escala cromática sin repetirse ninguna, de forma que se les da la misma importancia, se crean series derivadas, y se compone solamente con estas construcciones [5]. Esta forma de componer crea, pues, atonalidad pero mantiene una razón para la estructura que son las series.

Para componer con el método dodecafonista primero se ha de obtener, como hemos dicho anteriormente, una matriz donde la primera fila es la serie original (las 12 notas de la escala cromática en un orden) y la primera columna es la serie invertida (serie construida por movimiento contrario de la serie original, es decir, si la primera nota es un Do y la segunda un Re en la serie original, en la invertida será un Do y un Si). Si leemos la fila y columna al contrario (de izquierda a derecha o de abajo a arriba respectivamente) obtenemos la serie retrógrada y la serie retrógrada inversa en cada caso. El resto de elementos de la matriz se puede obtener transportando en base a los elementos de la primera fila o la primera columna.

En la figura 2.1 vemos un ejemplo. La primera fila de la matriz es la serie original (Do, Do♯, Re, Si, La♯, Sol♯, Mi, La, Re♯, Fa♯, Fa, Sol) y que la serie inversa se obtiene leyéndola de derecha a izquierda: Sol, Fa, Fa♯, Re♯, La, Mi, Sol♯, La♯, Si, Re, Do♯, Do). Para obtener la primera columna vemos que la distancia del primer elemento con el segundo es un semitono ascendente (de Do a Do♯), con lo que el segundo elemento de la serie inversa será un Si, es decir, un semitono descendente. Exactamente igual con el tercer elemento, en el que del segundo elemento al tercero de la serie original (Do♯y Re) hay otro semitono ascendente, con lo que en la serie invertida, será un semitono descendente respecto al segundo elemento de la serie inversa (Si), por lo que la serie por ahora sería: Do, Si, La♯. Si continuamos con este procedimiento obtenemos la serie inversa: Do, Si, La♯, Do♯Re, Mi, Sol♯, Re♯, La, Fa♯, Sol, Fa.

La numeración se basa en el número de la nota: 0 es Do, 1 es Do♯, 2 es Re... Y P es la serie original, R la retrógrada, I la inversa y RI es la retrógrada inversa. Para el resto de matriz, transportamos verticalmente o horizontalmente. Si lo hacemos horizontalmente, vemos que como del primer elemento al segundo hay un semitono descendente (de Do a Si), transportamos todos los elementos de la serie original un semitono hacia abajo. En la tercera fila

	I_0	I_1	I_2	I_{11}	I_{10}	I_8	I_4	I_9	I_3	I_6	I_5	I_7	
P_0	C	C♯/D♭	D	B	A♯/B♭	G♯/A♭	E	A	D♯/E♭	F♯/G♭	F	G	R_0
P_{11}	B	C	C♯/D♭	A♯/B♭	A	G	D♯/E♭	G♯/A♭	D	F	E	F♯/G♭	R_{11}
P_{10}	A♯/B♭	B	C	A	G♯/A♭	F♯/G♭	D	G	C♯/D♭	E	D♯/E♭	F	R_{10}
P_1	C♯/D♭	D	D♯/E♭	C	B	A	F	A♯/B♭	E	G	F♯/G♭	G♯/A♭	R_1
P_2	D	D♯/E♭	E	C♯/D♭	C	A♯/B♭	F♯/G♭	B	F	G♯/A♭	G	A	R_2
P_4	E	F	F♯/G♭	D♯/E♭	D	C	G♯/A♭	C♯/D♭	G	A♯/B♭	A	B	R_4
P_8	G♯/A♭	A	A♯/B♭	G	F♯/G♭	E	C	F	B	D	C♯/D♭	D♯/E♭	R_8
P_3	D♯/E♭	E	F	D	C♯/D♭	B	G	C	F♯/G♭	A	G♯/A♭	A♯/B♭	R_3
P_9	A	A♯/B♭	B	G♯/A♭	G	F	C♯/D♭	F♯/G♭	C	D♯/E♭	D	E	R_9
P_6	F♯/G♭	G	G♯/A♭	F	E	D	A♯/B♭	D♯/E♭	A	C	B	C♯/D♭	R_6
P_7	G	G♯/A♭	A	F♯/G♭	F	D♯/E♭	B	E	A♯/B♭	C♯/D♭	C	D	R_7
P_5	F	F♯/G♭	G	E	D♯/E♭	C♯/D♭	A	D	G♯/A♭	B	A♯/B♭	C	R_5
	RI_0	RI_1	RI_2	RI_{11}	RI_{10}	RI_8	RI_4	RI_9	RI_3	RI_6	RI_5	RI_7	

Figura 2.1: Matriz dodecafónica. [1]

se puede basar en la que acabamos de obtener o en la serie base, el transporte es diferente pero el resultado es el mismo.

> *Expuesto de esta manera, el método puede parecer arbitrario. Pero para Schönberg, se trataba de un modo sistemático de llevar a cabo lo que ya estaba haciendo en su música tonal: integrar la armonía y la melodía mediante la composición con un número limitado de conjuntos (aquí, los definidos por los segmentos de la serie), delimitar las frases y subfrases por medio de la saturación cromática (regulada por la aparición de las doce notas en cada exposición de la serie) y apoyarse en la variación en desarrollo. [6]*

2.2. OpenMusic

OpenMusic [7] es un entorno de programación visual orientado a objetos basado en Common Lisp. Se puede utilizar como entorno de programación visual para programación en Lisp. Implementa la mayoría de construcciones del lenguaje Common Lisp como condicionales, bucles, gestión de listas... Además de añadir un conjunto de estructuras y objetos musicales como acordes, secuencias de acordes, ritmos...

Incluye herramientas de reproducción MIDI, análisis de ficheros de sonido, representación de objetos 3D...

Fue diseñado en el IRCAM (Institut de Recherche et Coordination Acoustique/Musique) en 1998 y lanzado bajo licencia LGPL. Funciona bajo Windows, OS X y en los últimos meses se han empezado a publicar versiones para GNU/Linux.

Una lista de compositores que han reconocido utilizar OpenMusic son [2]:

- Alain Bancquart
- Brian Ferneyhough
- Joshua Fineberg
- Karim Haddad
- Eres Holz
- Michael Jarrell
- Fabien Lévy
- Fang Man
- Philippe Manoury
- Tristan Murail
- Kaija Saariaho
- Marco Stroppa

Además de poder programar de forma visual, al estar basado en Common Lisp, se permite crear funciones escritas en código y ser utilizadas en el entorno visual.

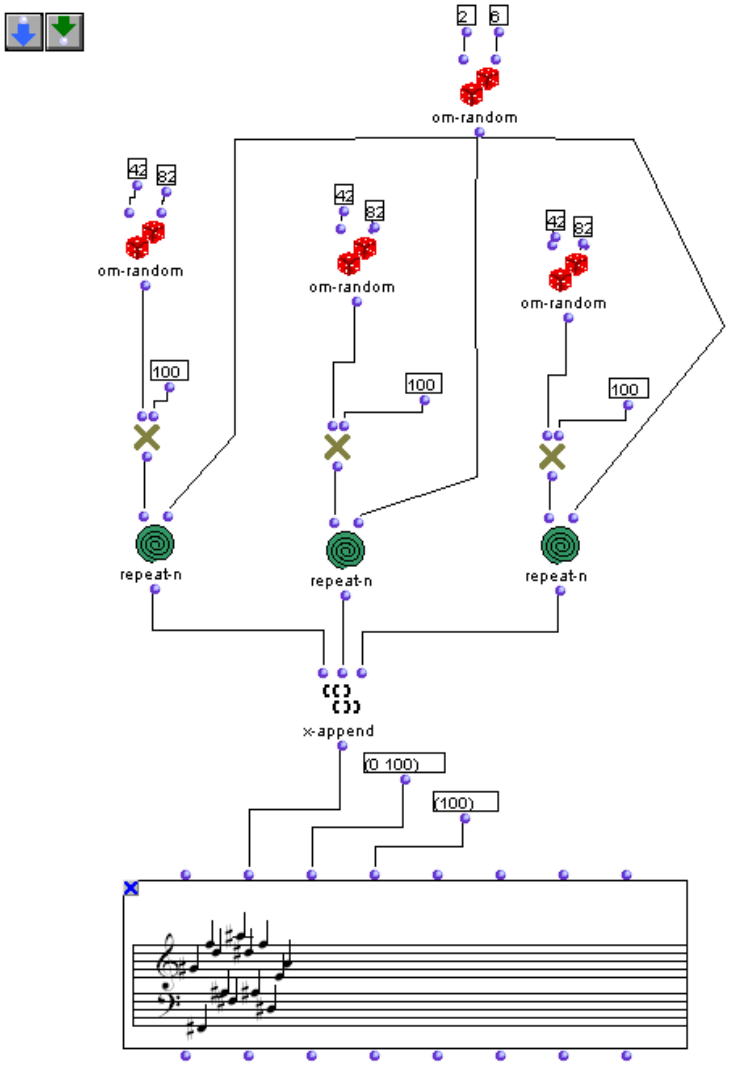

Figura 2.2: Ejemplo de programa escrito en OpenMusic. [2]

2.3. Lisp

Lisp es una familia de lenguajes de programación multiparadigma basados en listas y con notación basada en paréntesis. Fue creado por John McCarthy en 1958 en el MIT. Es el segundo lenguaje de alto nivel más viejo con alto uso hoy en día, FORTRAN es el primero. Los dialectos más conocidos de esta familia son Scheme y Common Lisp. Se popularizó en los entornos de la inteligencia artificial y fue pionero en conceptos de la ciencia de la computación como las estructuras de árboles o los tipos dinámicos.

El nombre de LISP deriva de "LISt Processing", dado que la estructura más importante del lenguaje es la lista (el código fuente de Lisp está compuesto por listas).[8]

La sintaxis del código se basa en expresiones S, donde el primer elemento después del paréntesis izquierdo es el nombre de la función y el resto de elementos son los argumentos. Por ejemplo:

```
(+ 1 3 (* 4 5))
; Output: 24
```

2.3.1. Common Lisp

Common Lisp es un dialecto de Lisp estandarizado por la ANSI creado para estandarizar las variantes de Lisp. Es un lenguaje de propósito general y multiparadigma: combina los paradigmas procedural, funcional y orientada a objetos. Al ser un lenguaje dinámico (los comportamientos del lenguaje se producen en tiempo de ejecución) facilita el desarrollo de software de forma incremental. Existen diferentes implementaciones del estándar, tanto de código abierto como propietarios. [9]

En comparación con el otro dialecto más famoso de Lisp, Scheme, las diferencias vienen dadas por algún cambio mínimo en la sintaxis (las declaraciones de funciones con defun dejan el nombre de la función fuera de la lista de argumentos en Common Lisp y en Scheme dentro de la lista, por ejemplo), soporte para diferentes tipos de números (Common Lisp soporta racionales, punto flotantes, enteros con precisión arbitraria... En cambio Scheme como mucho soporta números en coma flotante) y optimizaciones a nivel de la cola de recursión o del tamaño del código del lenguaje.

2.4. Python

Es un lenguaje de programación interpretado, multiparadigma (orientado a objetos, imperativo y funcional), tipado débil, multiplataforma y de código

abierto (Licencia PSFL). La filosofía de este lenguaje es la de una sintaxis preparada para hacer un código legible y sencillo.

2.5. QT

QT es un framework para aplicaciones multiplataforma utilizado para el desarrollo de interfaces gráficas en proyectos software o también programas sin interfaz para consola y consolas para servidores. También es software libre (bajo licencia GPLv3). El proyecto se hizo famoso en el desarrollo del entorno KDE y con el desarrollo de empresas como Nokia.

2.6. PyQt

La unión de los dos proyectos anteriores se conoce como PyQt, que aprovecha el conjunto de clases de Qt para crear una interfaz y las herramientas de diseño que incorpora y la simplicidad y posibilidad del uso de objetos del código de Python, además de ser un lenguaje que no necesita un ciclo de .^{ed}ición/compilación/linkeado/ejecución"que aumenta la productividad. Además, como las dos herramientas en las que está basado el proyecto es multiplataforma, lo convierte también en multiplataforma. Todo este conjunto de características hace que sea el complemento perfecto para crear interfaces y scripts rápidos para programas de OpenMusic.

Capítulo 3

Cuerpo del trabajo

El asistente de composición de música dodecafónica viene estructurado por 5 capas de software: la interfaz con Qt4, el script de generación de parámetros en Python3, el "patch"(en adelante programa) en OpenMusic (mayoritariamente código de Common Lisp) para la generación de la serie, el programa en OpenMusic para la generación de la partitura en OpenMusic y la visualización de la partitura completa en un visor externo (ya desarrollado, como puede ser Finale Notepad, que es gratuito y compatible con la versión del formato MusicXML que exporta OpenMusic).

La interfaz en Qt4 se ha diseñado utilizando Qt4 Designer, herramienta incluida en la instalación de Qt4 en el sistema y que permite utilizar todas las herramientas de Qt4 en un editor çlick and drop", pudiendo ver el resultado de la interfaz sin siquiera saber bajo qué código se va a ejecutar.

El fichero que se obtiene del editor Qt4 Designer es un .ui, que con la herramienta pyuic4 (Python-UI converter 4) convertimos el fichero .ui obtenido en un fichero de Python. Dado que la herramienta pyuic4 muestra por pantalla el resultado, se ha de ejecutar con el símbolo »"después del argumento del fichero a convertir para guardarlo en un fichero .py. El comando en Windows es (suponiendo que estamos en la carpeta con los 2 ficheros a utilizar):

```
pyuic4.bat RuleTranslator.ui > RuleTranslatorInterface.py
```

Este fichero es cargado por el script en Python y se le llama con las directivas propias de Qt4 en Python3. Carga la interfaz, convierte los valores de la interfaz en números y los agrupa en forma de lista antes de exportarlos a dos ficheros de texto para que OpenMusic los pueda abrir y procesar. En el código adjunto se puede ver claramente cómo funciona este sencillo script.

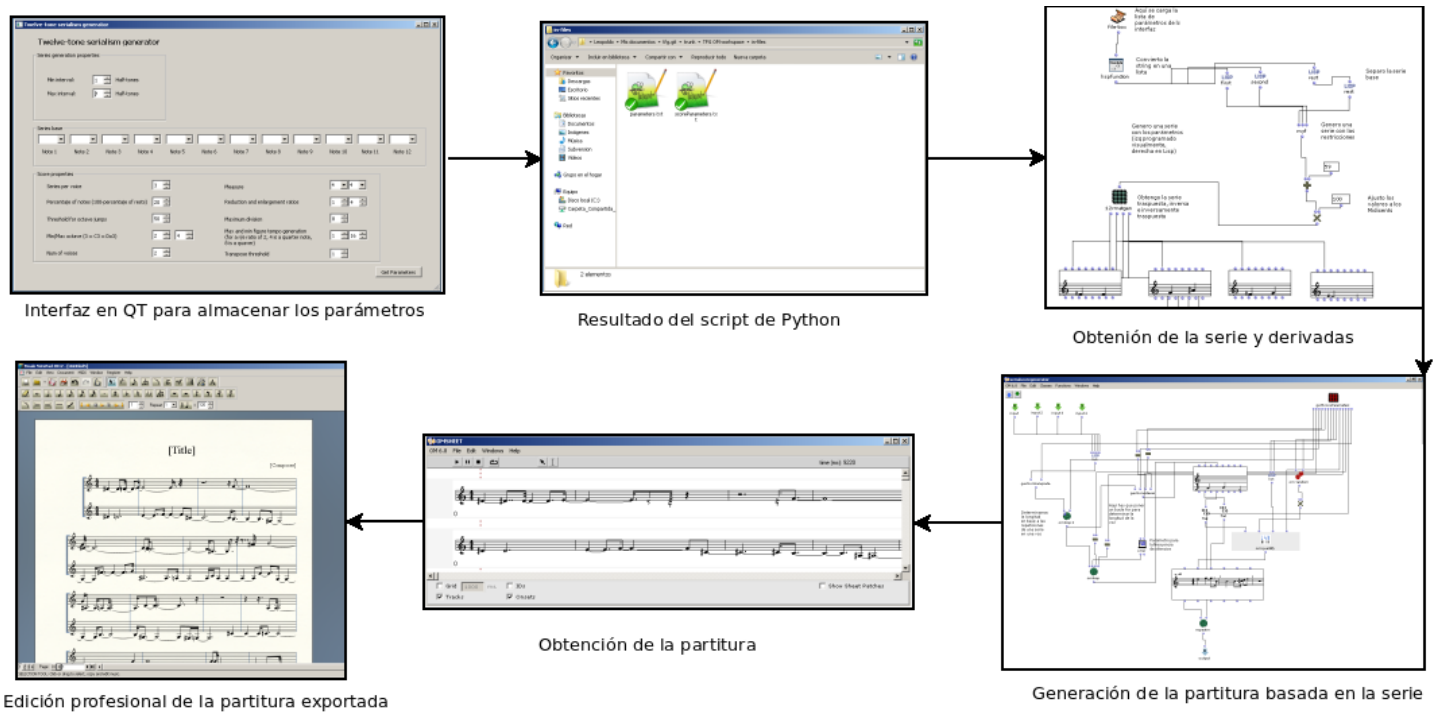

Figura 3.1: Diagrama del diseño del asistente.

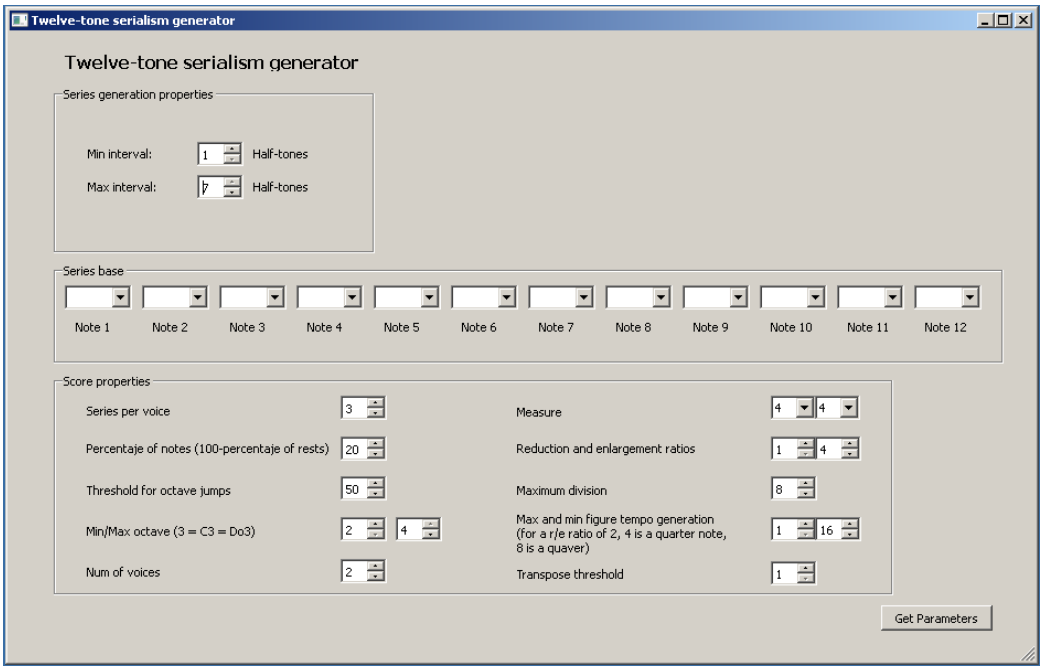

Figura 3.2: Interfaz en Qt bajo Windows 7

3.1. Generador de series

Una vez hemos obtenido los parámetros del compositor, se cargan con el método de lectura de ficheros de OpenMusic y se convierten con un método de Lisp llamado *read-from-string*. El método encargado de la generación de la serie basándose en las restricciones de mínimo y máximo salto de semitonos y en la serie semilla (introducidos por la interfaz) es el método escrito en Common Lisp *seriesGenerator* (empaquetado en forma de librería para hacerlo más accesible y reutilizable en otros proyectos).

Este código ha sido diseñado para obtener una solución siempre que sea posible dadas las restricciones utilizando backtracking en una implementación recursiva. El esquema del código es:

```
; Obtiene las notas que faltan en la secuencia parcial y las guarda en
    una lista llamada "notasfaltan"

; reordenar aleatoriamente notasfaltan

    ; para cada elemento de notasfaltan iteramos (bucle)
  ; Creamos una copia para no modificar la lista original
  ; seleccionar nueva nota y comprobamos si cumple las restricciones
  ; obtener posicion del primer cero (elemento a rellenar en la semilla)
  ; en la posicion -1 esta la nota anterior

  ; Si es la primera posicion de la lista no se mira el elemento
      anterior porque no hay
      ; llama recursivamente a seriesGenerator con la nueva lista
      ; si seriesGenerator devuelve una lista completa entonces termina
          el bucle y devuelve la lista


      ; Si no es la primera posicion de la lista, miramos las
          restricciones respecto a la nota anterior
          ; si las dos notas cumplen las restricciones, asigna la nueva
              nota en la posicion del primer cero (hueco en la semilla)
          ; llama recursivamente a seriesGenerator con la nueva lista
          ; si seriesGenerator devuelve una lista completa entonces
              termina y devuelve la lista



; si al terminar el bucle ninguna nota cumple la restriccion devuelve
    lista vacia
```

Para ver la implementación en Lisp, abrir la librería *twelvetonelib 1.0* dentro del Workspace, donde están los fuentes de todos los métodos utilizados.

Obtenida la serie con este código, procedemos a computar las 4 series básicas (sin trasporte) de la matriz dodecafónia: P, I, R, RI.

La serie P la obtenemos del método anterior y la inversa se obtiene aplicando el método *reverse* de Lisp. Para obtener la serie retrógrada programamos el mismo proceso que a mano.

Existen dos métodos en OpenMusic llamados x->dx y dx->x que devuelven una lista de intervalos de una lista de notas y viceversa. Es decir, si a x->dx se le pasan los valores 2000, 2500, 3200 devuelve (500 700), y si a dx->x le pasamos una nota (por ejemplo 1500) y los valores obtenidos con el ejemplo de antes (500 700), obtenemos (1500 2000 2700).

Si multiplicamos por -1 el valor de los intervalos (tal y como hacemos a mano, ya que si un intervalo es de X semitonos ascendente en la serie original, la inversa son X semitonos descendente) obtenemos la serie inversa y si le aplicamos el método *reverse* tenemos la retrógrada inversa.

Este proceso está programado en un patch de OpenMusic dado que visualmente es sencillo de programar. En el proyecto de OM se llama *12rmatgen*.

3.2. Generación de la partitura

En la generación de una partitura intervienen restricciones introducidas en la interfaz:

- número de series que aparecen en una voz (determinando en parte su longitud)
- porcentaje de notas (para generar también silencios)
- *threshold* para cambiar de octava (para saltar desde muy rápidamente a muy lentamente)
- octavas máximas y mínimas (desde Do2 hasta Do4)
- número de voces de la partitura, compás (2/4, 3/4, 4/4, 3/8, 6/8, 2/1...)
- máximo y mínimo valor para la reducción y ampliación (sirve para que el compositor pueda bloquear una voz ya generada y ver cómo puede quedar reduciendo o ampliando los tamaños de las notas)
- división máxima (siendo 8 la semicorchea, 4 la corchea, 2 la negra...)
- dimensión de las figuras máximas y mínimas (para un valor de factor de apliación y reducción de 2, 4 es una corchea, 8 es una negra...)
- un threshold para el transporte (a mayor valor, menos transporte se utiliza, es decir, más se utilizan las series básicas P, I, R, RI).

Capítulo 4

Conclusiones

Cuento cosas.

Bibliografía

[1] "Advanced music theory lesson 7- atonality and serialism." Website, Abril 2010. http://blog.indabamusic.com/2010/04/8160-advanced-music-theory-lesson-7-atonality-and-serialism/.

[2] Wikipedia, "Openmusic — wikipedia, the free encyclopedia," 2014. [Online; accessed 11-June-2014].

[3] A. Schoenberg, *Style and Idea*. University of California Press, 1984. http://books.google.es/books?id=jbXtxJezk5cC.

[4] A. Schoenberg, *Harmonielehre (Tratado de armonía)*. Real Musical, 1974. Copyright 1922 Universal Edition.

[5] Wikipedia, "Twelve-tone technique — wikipedia, the free encyclopedia," 2014. [Online; accessed 10-June-2014].

[6] J. P. Burkholder, D. J. Grout, and C. V. Palisca, *Historia de la música occidental*, vol. 1 y 2. Alicanza Música, 7 ed., 2008. Pages 894-906.

[7] IRCAM, "Openmusic," 2014.

[8] Wikipedia, "Lisp (programming language) — wikipedia, the free encyclopedia," 2014. [Online; accessed 12-June-2014].

[9] Wikipedia, "Common lisp — wikipedia, the free encyclopedia," 2014. [Online; accessed 12-June-2014].

Apéndice A

Más cosas

Aún faltan cosas por decir.