# DSLMusic

Dimitry Castex Marcelo Valdivia

4 de octubre de 2013

# Índice general

# Capítulo 1

# Libro Blanco

## 1.1. Introducción

DSLMusic, es un software que mediante un reconocedor, interprete y traductor de lenguaje, crea música a partir de sentencias gramaticales escritas por los usuarios. Este software posee los elementos básicos necesarios para crear melodías de nivel básico y medio, pues posee un panel de control que permite manipular la duración de un tono, modificar los intervalos entre sonidos (octavas), ingresar notas y silencios de acuerdo a las figuras musicales existentes (redonda, negra, blanca, corchea, etc.) y además, incluye un módulo que se encarga de exportar una melodía a pentagrama, dando la posibilidad de que un usuario externo pueda interpretar una composición usando el lenguaje universal de la música.

## 1.2. Justificacion

La propuesta se inserta dentro del perfil profesional que un Ingeniero en Computación de la Universidad de La Serena desarrolla durante su etapa de formación, específicamente bajo las características de un profesional capaz de adaptarse a la evolución de las tecnologías de la información, pues para llevar a cabo la realización del producto será necesario investigar acerca de librerías acordes al contexto de nuestro proyecto y usar técnicas de programación de acuerdo a estándares de la ingeniería de software.

Cabe la posibilidad que para los alumnos de este curso, el uso de una herramienta como ANTLR haya sido un tanto desligada de lo que se acostumbra a utilizar. Desde un comienzo se le ha dado énfasis al desarrollo de software acoplado solamente con el uso de base de datos y redes para algunos casos. Debido a esta inclinación generada hacia el desarrollo de software con las características antes mencionadas, se pasan por alto la realización de proyectos para la manipulación de lenguaje de datos, desarrollo de lenguajes de consulta para datos astronómicos, creación de compiladores, etc., proyectos que solo llegan a conocerse al momento de rendir el curso de Teoría de Autómatas y Lenguajes Formales.

Si bien ya nos encontramos en una etapa donde el curso se ha familiarizado con ANTLR y hasta se han creado aplicaciones Java que se acoplan perfectamente con las gramáticas generadas, el contexto de las aplicaciones construidas no se escapa de lo común y es un punto el cual hemos querido atacar.

Más que ir por el contrario a los proyectos de la Universidad de Columbia, la motivación

por crear un software que tuviese relación con el arte gráfico o la música era una idea que venía desde mucho antes y se tomó esta oportunidad para combinar una serie de contenidos dentro de un todo, hacer uso de aspectos de la ingeniería de Software, software libre, herramientas y librerías externas e incluirlos bajo un contexto fuera de lo común, para este caso en particular, la música.

Se espera que el desarrollar una aplicación de esta índole pueda ser el impulso inicial para que muchos estudiantes a futuro, puedan generar nuevas ideas de proyectos tomando en cuenta este y/o desarrollar nuevas versiones, tomando como base los aspectos de gestión de la configuración de la Ingeniería de Software

También se espera que se logre crear un impacto en la sociedad con la generación de aplicaciones ligadas al arte en general, fomentando así el desarrollo de la cultura.

## 1.3. Contexto

DSLMusic puede usarse por una persona o músico que necesite elaborar una partitura que él conoce, de forma rápida y limpia, la cual tenga directa relación con una obra vocal o instrumento solista (el software se limita a trabajar con un instrumento a la vez).

La ventaja principal que posee este software frente otros más sofisticados como "Finale"(software líder en el área de las aplicaciones para generar partituras), es que para obtener una simple partitura, requiere una menor curva de aprendizaje de los usuarios sobre el uso de la herramienta, pues no generaría mayor complicación a personas con conocimientos básicos de composición y un nivel novato en cuanto al uso de computadoras y software.

En cuanto a usuarios menos experimentados en el tema de la música, DSLMusic ofrece un panel de ayuda que permite escuchar una notas musicales antes de ser agregadas a una partitura, permitiendo la creación de melodías sin la necesidad de ser un experto en el tema.

## 1.4. Diseño

### 1.4.1. Diseño de la aplicación

DSLMusic en una aplicación de escritorio destinada a la creación de melodías y exportación de partituras a partir de sentencias gramaticales.

Un usuario inexperto en temas relacionados con la música, podrá hacer uso de esta aplicación sin problemas, pues posee un pequeño piano (que hace una muestra del sonido antes de ser incluido dentro de una melodía), un panel de control para duración y tonalidad de las notas musicales y además, una interfaz gráfica de usuario bastante intuitiva.

Primeramente, los usuario tendrán que componer melodías a partir de sentencias gramaticales las cuales pertenecen a un lenguaje propio de la aplicación. Éstas deberán escribirse en el panel de composición de melodías y luego, si no hay errores en su traducción, se le podrá dar inicio a la reproducción de la composición.

Si el usuario lo desea, una composición escrita de una forma específica podrá ser exportada a partitura, la cual podría ser vista por cualquier persona que toque algún instrumento como una flauta o piano y luego podrá interpretarla y obtener el mismo resultado mostrado por la aplicación.

### 1.4.2. Génesis de la aplicación

#### Theremin

El theremín (théremin o théreminvox), llamado también eterófono en su versión primitiva, es uno de los primeros instrumentos musicales electrónicos, inventado en el año 1919 por el físico y músico ruso Lev Serguéievich Termen.

El diseño clásico consiste en una caja con dos antenas, en las cuales, al momento de pasar una mano por encima de cada una de las antenas, un sonido es ejecutado. La antena derecha suele ser recta y en vertical, y sirve para controlar la frecuencia o tono (mientras más cerca esté la mano derecha de la misma, más agudo será el sonido producido). La antena izquierda es horizontal y con forma de bucle, y sirve para controlar el volumen (mientras más cerca de la misma esté la mano izquierda de la antena, más baja el volumen). Originalmente, su versión más primitiva fue llamada Aetherophone (que se podría traducir como Eterófono), y poseía sólo la antena para el control de tono. Dicho diseño fue tempranamente mejorado por el inventor, añadiendo posteriormente la antena para controlar el volumen. Actualmente, algunos de los modelos caseros y comercializados de Theremin disponen tan sólo de la antena que controla el tono, lo cual siendo rigurosos los convierte en realidad en un .ᴱᵗerófonoz su uso frecuentemente es el de un aparato para efectos especiales más que un instrumento musical, al no poder acentuar ni separar las notas producidas.

#### Theremin virtual y la generación de Sonido

La forma de generar el sonido mediante un theremin solo haciendo uso de las manos para generar notas unas tras otras y a su vez haciendo representación de silencios, fue la motivación suficiente para que Jorge Rubira, un programador dedicado a hacer videotutoriales, comenzara a trabajar en la construcción de un theremin virtual, en el cual mediante el uso del mouse al arrastrarlo por sobre un panel creado en java, crea notas de forma constante, las cuales a través de arreglos de bytes, genera un buffer y es liberado a través de la tarjeta de sonido.

El sonido se produce cuando hay una variación en la presión dentro de un medio. Esta variación es audible cuando el movimiento se produce dentro de un rango de velocidad determinada. Por ejemplo, una señal continua, aunque existe "presión", no es audible por no existir fluctuaciones. Por ello, cuando se muestrea y se visualiza un sonido suele ser una señal oscilante.

Cuando esta oscilación se realiza a una frecuencia determinada constante, no solo se emite un sonido, si no que se emite un tono. La variación de la frecuencia de la señal emitida provoca la modificación del tono. Para la creación de notas musicales se han estipulado frecuencias concretas siendo la referencia la nota LA (A) pudiendose obtener a partir de la siguiente fórmula $55Hz * 2^{octava}$: $55Hz$, $110Hz$, $220Hz$, $440Hz$, $880Hz$, etc (contando las octavas en base 0).

El resto de notas siguen una función exponencial con la fórmula $frecuencia = 55Hz \cdot 2^{octava+(nota/12)}$, igualmente asumiendo que la numeración de las octavas y notas están en base 0, es decir la octava 1 es la 0 y la nota A es la 0.

### Muestreo de sonido

En cuanto al muestreo del sonido hay que tener en cuenta diferentes parámetros de configuración. Uno de estos parámetros es la cantidad de canales que se desea muestrear concurrentemente. Normalmente suelen utilizarse dos modos: 1 canal (mono) y 2 canales (estéreo). Otro parámetro a configurar es el número de bits que representa la precisión de cada muestra En la actualidad lo habitual es 8bits, 16bits y 32bits. Finalmente, la frecuencia de muestreo representa cada cuanto tiempo tomaremos una muestra. Lo habitual es 11.025Hz, 22.050Hz y 44.100Hz. La configuración preferencial para crear un teremin virtual será con un muestreo en mono (solo 1 canal), 8bits que se reflejará en el tipo de variable a utilizar (byte) y 22.000Hz que afectará a la cantidad de bytes que se introducirán en el buffer de salida en 1 milisegundo. Las clases que permiten ejecutar el teremin virtual se encuentran en la sección de anexos.

### Emisión de sonido en Java

Debido a que el sonido del theremin virtual será generado a través del buffer de salida de sonido, no será necesario tener sonido precargados ni hacer uso de librerías. Para llevarlo a cabo, será necesario crear un oscilador que genere una señal sinusoidal en tiempo real. Para emitir sonidos hay que abrir un buffer de salida al dispositivo de audio y enviar vectores de bytes que se ejecutarán según la configuración determinada. El theremin virtual creado por Jorge Rubira consiste en una ventana la cual detectará la posición del puntero del mouse para generar el sonido. Si el mouse es movido hacia arriba y abajo, el volumen del sonido subirá y bajará. Si el mouse es movido hacia la izquierda o derecha, la frecuencia de la señal emitida variará y será posible generar el sonido (distintas notas musicales).

### Frecuencias para notas y octavas

Si bien en un comienzo fue una opción generar música a partir de imágenes (representar cada pixel como un sonido en particular), pareció ser más interesante la idea de componer música que en realidad tuviese sentido.

Se tomó como base el funcionamiento del theremin virtual, usando el buffer de salida de sonido para crear tonos.

A partir de una señal sinusoidal (función seno de la librería propia de java Math.sin), el sonido emitido con esta función será similar al tono que escuchamos al levantar un teléfono. Sin embargo, no todas las señales que tienen frecuencias deben ser sinusoidales. Existen señales cuadradas, en sierra y otras más. Cuanto más brusca o más fluctuaciones tiene una señal, más “agresivo” es el sonido que genera.

Luego de trabajar con ondas sinusoidales, es necesario entender como se genera cada nota musical a partir de las frecuencias.

El oído humano capta sólamente frecuencias que estén por encima de los 20Hz y por debajo de los 20.000 (muy aproximadamente). Así pues, y con mucha suerte, sólo podemos oir unas 10 octavas como mucho, con doce notas cada una.

Con octava nos referimos a un intervalo que separa dos sonidos cuyas frecuencias fundamentales tienen una relación de dos a uno.

La nota La sirve como referencia para todas las demás. A menudo se denomina ”nota de afinar”. Se produce un La de afinar cuando el aire vibra 440 veces por segundo, es decir a 440 hertzios. Por convención, a la octava que contiene esta nota La se le suele considerar la cuarta.

Hay otra nota La, de una .ºctava"superior (la quinta octava) cuando el aire vibra a 880 hertzios, y otra más cuando vibra a $880 \cdot 2$ (sexta octava), y otra a $880 \cdot 2 \cdot 2$ (séptima octava), etc. (haciendo mención a la relación dos a uno antes mencionada), del mismo modo que hay un La que se produce cuando el aire vibra a $\frac{440}{2}$ (tercera octava) y otra a $\frac{\frac{440}{2}}{2}$ (segunda octava).

Para hallar la frecuencia de una nota cualquiera mediante una expresión matemática, se suele coger una frecuencia de referencia, por ejemplo el La de afinar (La4, 440 Hertzios) y se multiplica por la raiz duodécima de 2 elevado al número de semitonos que separa el la de afinar de la nota que estamos buscando.

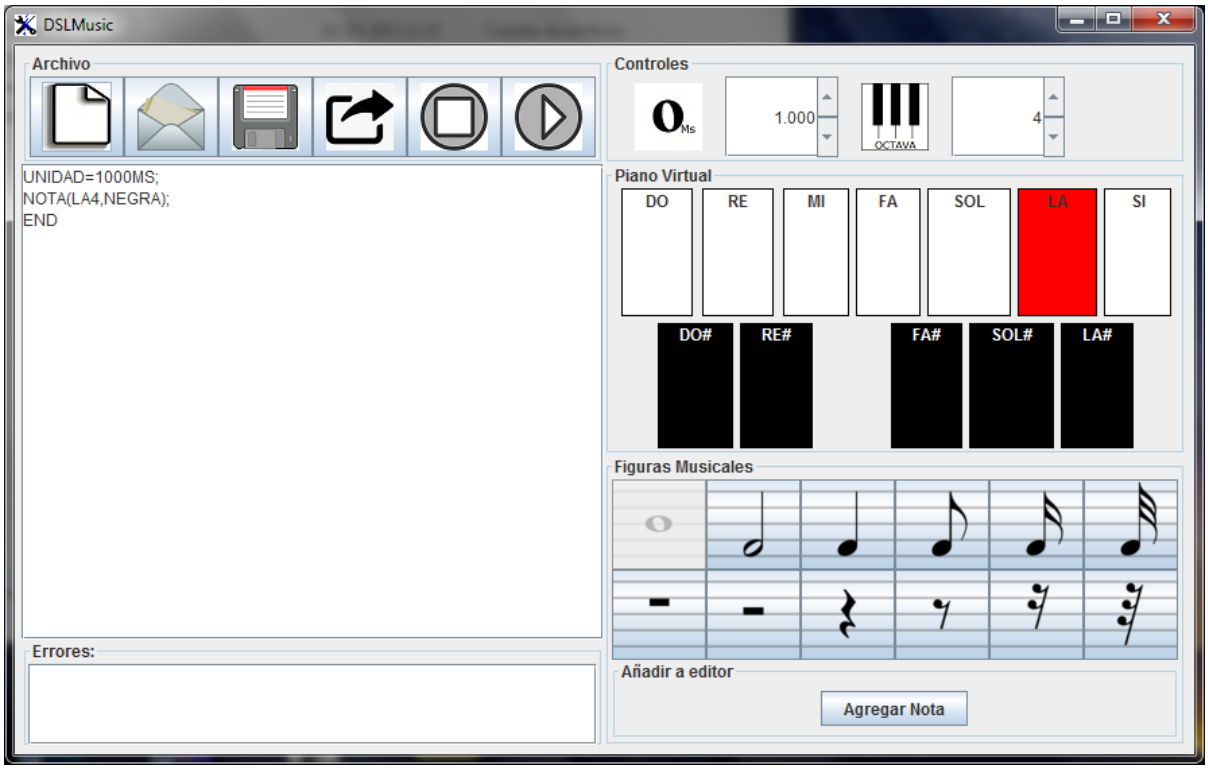
Esta expresión puede ser difícil de codificar en algunos lenguajes de programación, ya que es muy probable que no dispongan de funciones matemáticas para hallar una raíz duodécima. Adaptarla un poco es muy sencillo, ya que la raíz duodécima de 2 se puede calcular como 2 elevado a $\frac{1}{12}$ , con lo que la expresión quedaría de ésta manera:

$$f(n, o) = 440 \cdot 2^{(o-4)+\frac{n-10}{12}}$$

Utilizando esta fórmula, es posible obtener las frecuencias para cada nota musical en estado normal, sostenida y con la octava que deseamos. A modo de ejemplo, listamos las frecuencias más utilizadas para representar sonidos los cuales son comunes para el oído humano:

| | | | |
|---|---|---|---|
| $Do_4 : 261,625565Hz$ | $Do_5 : 523,251131Hz$ | $Do_6 : 1046,502261Hz$ | $Do_7 : 2093,004522Hz$ |
| $Do\sharp_4 : 277,182631Hz$ | $Do\sharp_5 : 554,365262Hz$ | $Do\sharp_6 : 1108,730524Hz$ | $Do\sharp_7 : 2217,461048Hz$ |
| $Re_4 : 293,664768Hz$ | $Re_5 : 587,329536Hz$ | $Re_6 : 1174,659072Hz$ | $Re_7 : 2349,318143Hz$ |
| $Re\sharp_4 : 311,126984Hz$ | $Re\sharp_5 : 622,253967Hz$ | $Re\sharp_6 : 1244,507935Hz$ | $Re\sharp_7 : 2489,015870Hz$ |
| $Mi_4 : 329,627557Hz$ | $Mi_5 : 659,255114Hz$ | $Mi_6 : 1318,510228Hz$ | $Mi_7 : 2637,020455Hz$ |
| $Fa_4 : 349,228231Hz$ | $Fa_5 : 698,456463Hz$ | $Fa_6 : 1396,912926Hz$ | $Fa_7 : 2793,825851Hz$ |
| $Fa\sharp_4 : 369,994423Hz$ | $Fa\sharp_5 : 739,988845Hz$ | $Fa\sharp_6 : 1479,977691Hz$ | $Fa\sharp_7 : 2959,955382Hz$ |
| $Sol_4 : 391,995436Hz$ | $Sol_5 : 783,990872Hz$ | $Sol_6 : 1567,981744Hz$ | $Sol_7 : 3135,963488Hz$ |
| $Sol\sharp_4 : 415,304698Hz$ | $Sol\sharp_5 : 830,609395Hz$ | $Sol\sharp_6 : 1661,218790Hz$ | $Sol\sharp_7 : 3322,437581Hz$ |
| $La_4 : 440,000000Hz$ | $La_5 : 880,000000Hz$ | $La_6 : 1760,000000Hz$ | $La_7 : 3520,000000Hz$ |
| $La\sharp_4 : 466,163762Hz$ | $La\sharp_5 : 932,327523Hz$ | $La\sharp_6 : 1864,655046Hz$ | $La\sharp_7 : 3729,310092Hz$ |
| $Si_4 : 493,883301Hz$ | $Si_5 : 987,766603Hz$ | $Si_6 : 1975,533205Hz$ | $Si_7 : 3951,066410Hz$ |

## 1.5. Ejemplo visual




## 1.6. Descripción del lenguaje

DSLMusic posee un lenguaje propio pero con un toque de algunos lenguajes como los vistos a lo largo de nuestra formación en la Universidad como lo son Pascal, C, C♯ y el más usado Java, pues las sentencias gramaticales que dan vida a las melodías son similares a la declaración de funciones o más bien llamados métodos, los cuales reciben por parámetro una serie de valores y Strings. Además, cada sentencia del tipo mencionado, debe ser finalizada por medio de la puntuación coma ”,”.

La estructura del lenguaje será vista con más detalle en el manual del lenguaje para la aplicación.

# Capítulo 2

# Tutorial del lenguaje

## 2.1. Instalación

DSLMusic es una aplicación que en un solo archivo trae empaquetado todo lo necesario para su funcionamiento (clases propias y librerías externas). Los usuarios que deseen ejecutar la aplicación, solo deberán comprobar que tienen instalada la máquina virtual de java. Si DSLMusic no puede ejecutarse, el usuario tendrá que descargar la última versión de java desde el siguiente sitio:

`http://www.java.com/es/download/`

## 2.2. Codificación del programa

DSLMusic fue codificado tomando en cuenta aspectos importantes de la ingeniería de software como la mantenibilidad. Esta etapa del ciclo de un software es la de mayor coste pues para lograr aspectos de mantención a futuro es necesario que el desarrollo del software mantenga un enfoque de calidad en todo momento, creando en este caso una aplicación basada en componentes y haciendo uso de la gestión de la configuración.

El código fuente crea principalmente 3 tipos de objetos para su funcionamiento, comenzando por una "Nota", la cual es finalmente representada como una frecuencia que genera una nota musical. Junto a esto, se tiene un objeto del tipo "Partitura", una especie de contenedor que aloja objetos del tipo "Nota"para luego ser reproducidos como una melodía. El tercer tipo de objeto es "Pentagrama Gráfico.[el] cual se encarga de exportar las sentencias gramaticales escritas por un usuario a un pentagrama que contiene las figuras musicales (si las sentencias están mal escritas, el pentagrama no podrá ser generado).

### 2.2.1. Definicion de objetos

#### Objeto Nota

Un objeto de tipo Nota recibe dos parámetros, uno para la frecuencia y otro para la duración.

```
public Nota(double frec,double dur){
```

```
    frecuencia = frec;
    duracion = dur*22;
    b=new byte[(int)duracion]; //Buffer
    for (int n=0;n < b .length;n++){
      b[n]=(byte)(Math.sin(frecuencia*n*Math.PI*2/22000)*127);
    }
}
```

Con los parámetros recibidos, un arreglo de bytes es enviado al buffer de sonido y una nota es emitida.

#### Objeto Partitura

Como se explicó anteriormente, un objeto Partitura es un contenedor de Notas.

```
public Partitura(){
    arregloNotas = new ArrayList<Nota>();
}
public void addNota(double frecuencia,double duracion){
    Nota n = new Nota(frecuencia,duracion);
    arregloNotas.add(n);
}
public void addSilencio(double duracion){
    Nota n = new Nota(0,duracion);
    arregloNotas.add(n);
}
```

#### Objeto Pentagrama Gráfico

Para generar un pentagrama a partir de sentencias gramaticales, este objeto recibe un String y usa la librería externa llamada ABC4J para parsearlo y mostrar en pantalla el pentagrama con las figuras musicales.

```
public PentagramaGrafico(String notas){
    Tune tune = new TuneParser().parse(notas);
    JScoreComponent scoreUI =new JScoreComponent();
    scoreUI.setTune(tune);
}
```

De esta forma, una vez que ANTLR detecte una sentencia gramatical correcta, ésta será interpretada y traducida a una sentencia aceptada por la librería y será parseada para mostrar en forma gráfica los resultados.

## 2.3. Uso de DSLMusic para la creación de una melodía

DSLMusic posee dos formas de composición de melodías, una que acepta como parámetro la nota musical octava y figura músical, y otra que solo necesita los parámetros de frecuencia y duración del sonido a emitir.

### 2.3.1. Parámetros de nota, octava y figura musical

Para generar una melodía a través de este método, es necesario hacer una composición de la siguiente manera:

```
UNIDAD=1000MS;
NOTA(LA4,REDONDA);
NOTA(SOLS4,REDONDA);
SILENCIO(NEGRA);
END
```

Primero, con UNIDAD se declara la unidad de tiempo en segundos o milisegundos para la duración de una figura musical redonda". A modo de ejemplo, si una figura musical vale 1000MS, una figura blanca durará 500MS y una negra 250MS.

Para añadir notas o silencios a una partitura, se hace la declaración en este caso NOTA(LA4,REDONDA); en donde, primero se especifica al comienzo de la sentencia si nos encontramos con una nota o un silencio, seguido de una entrada de parámetros donde se recibe la nota seguido del valor de la octava (intervalo para separar sonidos) y luego en el siguiente parámetro, se especifica la figura musical, para este caso una redonda.

Para finalizar la melodía, se agrega la sentencia END y si no hay errores de sintaxis, la composición estaría lista para ser reproducida.

### 2.3.2. Parámetros de frecuencia y duracion

Para generar una melodía a través de este método, es necesario hacer una composición de la siguiente manera:

```
SONIDO(440HZ,1000MS);
SILENCIO(10MS);
END
```

A diferencia del método anterior, aquí no es necesario declarar la unidad de tiempo al comienzo. Para agregar un sonido (como se ingresa en las frecuencias que uno desee, ya no los consideramos como notas) , tomando como ejemplo la sintaxis SONIDO(440HZ,1000MS); se aprecia que primeramente se hace la distinción entre un silencio y un sonido. Luego, como primer parámetro se espera una frecuencia seguido de su medida en HZ. El último parámetro recibe la duración del sonido, expresado en milisegundos. Para finalizar la composición, se añade la sentencia END y ya puede ser reproducida.

Se debe resaltar que para generar un pentagrama con figuras musicales, este método de composición no es el indicado, puesto que no se hace representación de notas exactas y menos de figuras musicales. Para exportar a representación gráfica es estrictamente necesario componer basándose en el primer método mencionado.

# Capítulo 3

# Manual del lenguaje

## 3.1. Convención léxica

### 3.1.1. Tokens

Los tokens de nuestro lenguaje son las notas musicales, las figuras musicales, número (que representa tiempo, octavas), unidad de frecuencia y unidad de tiempo.

Notas musicales: DO,RE,MI,FA,SOL,LA,SI
Notas musicales (semitonos): DOS,RES,FAS,SOLS,LAS
Figuras musicales: REDONDA,BLANCA,NEGRA,CORCHEA,SEMICORCHEA,FUSA
Tiempo: número entero positivo generado a partir de tokens del 0 al 9
Unidad de frecuencia: HZ
Unidad de tiempo: MS

### 3.1.2. Espacios en blanco, tabulaciones y saltos de línea

DSLMusic no toma en cuenta a estos tipos de caracteres, pues al momento hacer la interpretación de una composición, se usa el canal oculto de ANTLR channel-hidden y de esta forma, por cada espacio, tabulación o salto de línea, la ejecución de la aplicación no se ve afectada, los trata como nulo, como si no existiesen.

### 3.1.3. Palabras Clave

DSLMusic posee palabras claves que indican el método que se está ocupando para crear una composición.

#### Palabras Clave Método 1: nota,octava y figura musical

UNIDAD=: con esta sentencia, se indica la cantidad de tiempo asignada a la figura musical redonda. Si la sentencia se encuentra en el panel de composición, se da por entendido que el usuario trabajará con el método 1 en donde se usa una nota, octava y figura musical para crear las melodías.

NOTA(nota octava, figuraMusical): con esta palabra clave, se indica que se quiere añadir una nota musical a una partitura, la cual dentro de sus parámetros espera la especificación

de la nota musical, la octava y la figura musical. Se debe respetar la sintaxis mostrada, los paréntesis, clemillas y punto y coma.

SILENCIO(figuraMusical): esta sentencia permite añadir a la partitura un silencio entre notas, la cual recibe dentro de su parámetro una figura musical.

END: esta palabra clave permite dar la finalización a una composición.

**Palabras Clave método 2: frecuencia y duración**

SONIDO('frecuencia','tiempo'): con esta palabra clave, podemos agregar un sonido dentro de un objeto partitura. Obteniendo por parámetro la frecuencia y el tiempo, se añade un arreglo de bytes al buffer de salida de sonido y sonará la composición hecha por el usuario.

SILENCIO('tiempo'): esta sentencia permite a los usuarios que utilicen el método 2 para componer, que puedan añadir silencios a un objeto partitura solo especificando el tiempo en segundos o milisegundos.

'END': esta palabra clave permite dar la finalización a una composición.

# Capítulo 4

# Plan de Proyecto

## 4.1. Objetivo General

Crear un DSL para la composición de melodías e implementarlo en un software para la generación de música y exportación a pentagrama

## 4.2. Objetivos Específicos

### 4.2.1. Investigación y análisis de la teoría musical

- Investigación y estudio sobre las frecuencias involucradas en notas musicales.
- Investigación y estudio sobre el comportamiento de las figuras musicales en una melodía
- Investigación y estudio sobre intervalos entre sonidos (octavas)

### 4.2.2. Estudio de Factibilidad

- Investigación sobre el instrumento Theremin (funcionamiento)
- Investigación y estudio sobre la representación de sonidos mediante la introducción de bytes al flujo de salida
- Análisis de Theremin Virtual creado por Jorge Rubira
- Investigación y estudio de librerías para la creación de partituras

### 4.2.3. Diseño e implementación

**DSL**

- Diseñar DSL para la composición de melodías
- Implementar DSL en una gramática utilizando ANTLR
- Verificar que la gramática implementada se encuentre en Forma Normal de Greibach

- Realizar pruebas de unidad sobre las reglas y producciones de la gramática generada

**Producto de Software**

- Implementar módulo para el control del buffer de salida de sonido
- Implementar módulo para de interpretación de notas musicales a arrays de bytes
- Implementar módulo para la inserción de notas musicales a partitura
- Implementar módulo para interpretación de sentencias gramaticales y exportación a sonido y partitura gráfica
- Implementar interfaz de usuario

## 4.3. Ciclo de Vida Adoptado

El ciclo de vida elegido para este proyecto es Incremental - Evolutivo debido a las características del problema:

- Los requisitos cambian conforme al desarrollo del producto final.
- Debido a las fechas impuestas por término de semestre, es necesario introducir una versión limitada y funcional del producto.
- Los requisitos centrales están bien definidos y las extensiones del producto aún no son totalmente definidas.
- Para obtener un producto final, es necesario un mayor número de iteraciones, motivación principal para quienes rindan el curso a futuro y puedan generar nuevas versiones del software y satisfacer nuevos requisitos.
- La cantidad de desarrolladores es baja, no es suficiente para hacer un software sofisticado en el tiempo dado, se prioriza los requerimientos que permiten desarrollar el núcleo del software.

## 4.4. Planificación

### 4.4.1. Planificación general

| ID | TIPO | TAREAS CONCRETAS | INICIO | TERMINO |
|---|---|---|---|---|
| 1 | Planificación | Definir, delimitar y concretar idea de proyecto | 09-07-2013 | 15-07-2013 |
| 2 | Documentación | Plan de proyecto y ciclo de vida | 16-07-2013 | 19-07-2013 |
| 3 | Documentación | Definición de requerimientos | 22-07-2013 | 31-07-2013 |
| 4 | Documentación | Diseño | 05-08-2013 | 22-08-2013 |
| 5 | Componente | Construcción de gramática | 26-08-2013 | 02-09-2013 |
| 6 | Componente | Traductor de frecuencias | 03-09-2013 | 09-09-2013 |
| 7 | Componente | Traductor ABC4J | 10-09-2013 | 16-09-2013 |
| 8 | Componente | Definir acciones en ANTLR | 17-09-2013 | 20-09-2013 |
| 9 | Componente | Entradas: eventos de usuario | 23-09-2013 | 24-09-2013 |
| 10 | Componente | Interfaz gráfica de usuario | 25-09-2013 | 27-09-2013 |
| 11 | Implementación | Integración de componentes | 30-09-2013 | 02-10-2013 |
| 12 | Calidad de SW | Pruebas | 03-10-2013 | 03-10-2013 |

### 4.4.2. Planificación en detalle y responsables

| ID | TAREAS | INICIO | TERMINO | RESPONSABLE |
|---|---|---|---|---|
| 1 | Definir idea de proyecto | 09-07-2013 | 11-07-2013 | Castex |
| | Acotar el problema | 12-07-2013 | 13-07-2013 | Valdivia |
| | Factibilidad técnica | 14-07-2013 | 15-07-2013 | Castex y Valdivia |
| 2 | Definicion de objetivos | 16-07-2013 | 17-07-2013 | Castex y Valdivia |
| | Caracterización del problema | 17-07-2013 | 17-07-2013 | Castex y Valdivia |
| | Elección de ciclo de vida de SW | 18-07-2013 | 19-07-2013 | Castex y Valdivia |
| 3 | Captura de requerimientos | 22-07-2013 | 24-07-2013 | Castex y Valdivia |
| | Definición de requerimientos funcionales | 25-07-2013 | 27-07-2013 | Castex y Valdivia |
| | Definición de requerimientos no funcionales | 28-07-2013 | 31-07-2013 | Castex y Valdivia |
| 4 | Diseño general | 05-08-2013 | 15-08-2013 | Castex y Valdivia |
| | Diseño DFD | 16-08-2013 | 22-08-2013 | Castex y Valdivia |
| 5 | Gramática para primer método de composición | 26-08-2013 | 29-08-2013 | Castex y Valdivia |
| | Gramática para segundo método de composición | 30-08-2013 | 30-08-2013 | Castex y Valdivia |
| | Unión de gramáticas | 31-08-2013 | 31-08-2013 | Castex y Valdivia |
| | Comprobar forma normal en la gramática | 01-09-2013 | 02-09-2013 | Castex y Valdivia |
| 6 | Traducir nota y ocatava a frecuencia | 03-09-2013 | 03-09-2013 | Castex y Valdivia |
| | Traducir figura musical a duración en milisegundos | 04-09-2013 | 04-09-2013 | Castex y Valdivia |
| | Traducir una frecuencia y duración a arreglos de bytes | 05-07-2009 | 07-09-2013 | Castex y Valdivia |
| | Enviar arreglos de bytes al buffer de salida | 08-09-2013 | 09-09-2013 | Castex y Valdivia |
| 7 | Traducir nota, ocatava y figura musical a notación ABC4J | 10-09-2013 | 13-09-2013 | Castex y Valdivia |
| | Formatear notación para ingresarla a librería | 13-09-2013 | 14-09-2013 | Castex y Valdivia |
| | Generar pentagrama gráfico | 15-09-2013 | 16-09-2013 | Castex y Valdivia |
| 8 | Implementar acciones para composición método 1 | 17-09-2013 | 18-09-2013 | Castex y Valdivia |
| | Implementar acciones para composición método 2 | 19-09-2013 | 20-09-2013 | Castex y Valdivia |
| 9 | Implementar entrada por archivo | 23-09-2013 | 23-09-2013 | Castex y Valdivia |
| | Implementar entrada estándar por teclado | 24-09-2013 | 24-09-2013 | Castex y Valdivia |
| 10 | Panel de composición y captura de errores | 25-09-2013 | 25-09-2013 | Valdivia |
| | Panel de control para composición | 26-09-2013 | 27-09-2013 | Valdivia |
| | Panel de ayuda: piano virtual | 25-09-2013 | 25-09-2013 | Castex |
| | Panel de archivo | 26-09-2013 | 27-09-2013 | Castex |
| 11 | Integrar módulos de traducción | 30-09-2013 | 30-09-2013 | Castex y Valdivia |
| | Integrar módulos de entradas e interfaz de usuario | 01-10-2013 | 01-10-2013 | Castex y Valdivia |
| | Integración final | 02-10-2013 | 02-10-2013 | Castex y Valdivia |
| 12 | Junit | 03-10-2013 | 03-10-2013 | Castex y Valdivia |

## 4.5. Ambiente de desarrollo

El código fue desarrollado bajo el sistema operativo Windows 7, haciendo uso del IDE Eclipse en su versión 3.7 Indigo. La aplicación es multiplataforma, desarrollada en lenguaje Java con JDK versión 1.7 y ejecutada gracias en cualquier sistema operativo que tenga instalada la máquina virtual de Java.

## 4.6. Convención de estilos de codificación

### 4.6.1. ANTLR

- Separación uniforme e identación
- Nombres lógicos y uniformes para las variable
- Escribir código en orden lógico
- Comentar aquellos bloques de código difíciles de entender su funcionalidad a simple vista

### 4.6.2. Java

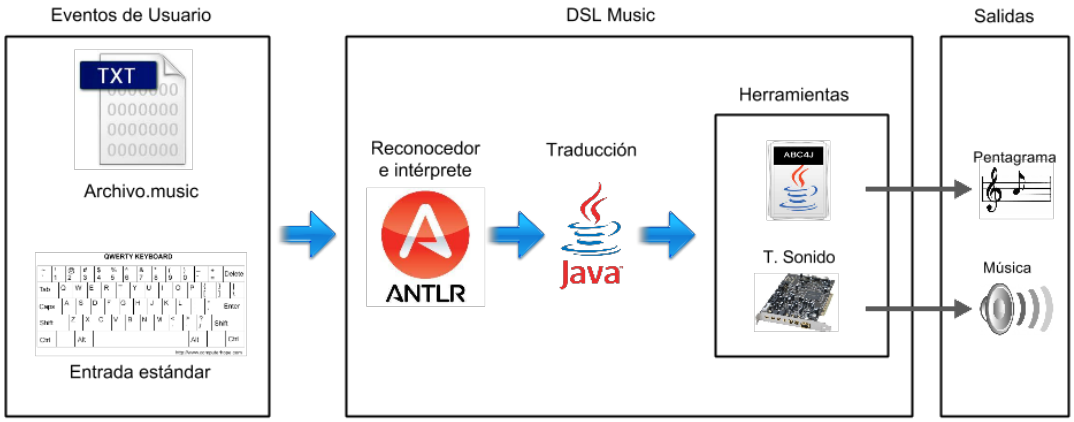
Se ha adoptado la convención estándar para la programación en lenguaje Java. Ésta se encuentra disponible en el sitio oficial de Oracle.

`http://www.oracle.com/technetwork/java/codeconv-138413.html`

# Capítulo 5

# Diseño Arquitectónico

## 5.1. Diseño General




### 5.1.1. Eventos de usuario

Los eventos de usuario son generados a partir de dos entradas, una por archivo de texto de extensión ".musicz otra mediante la entrada estándar de teclado. Se requiere que la entrada corresponda a un conjunto de sentencias gramaticales definidas para el DSL construído.

### 5.1.2. DSL Music

DSL Music una vez escuchada la entrada de usuario, procede a utilizar una serie de herramientas antes de generar las salidas esperadas.

#### Reconocedor e intérprete

DSL Music para reconocer e interpretar las sentencias gramaticales de las entradas de usuario, usa ANTLR. Con esta herramienta podemos parsear una serie de sentencias ingresadas por el usuario para luego interpretarlas y generar al mismo tiempo alguna acción, en este caso, componer melodías.

#### Traducción

Una vez que es comprobado que la entrada de usuario fue correcta mediante el uso de ANTLR, será necesario crear un parseador interno en Java que permita por un lado, traducir una sentencia del tipo "(NOTAOCTAVA,FIGURAMUSICAL)" a una frecuencía en medida en hertz y una duración en milisegundos. Por otro lado, fue necesario generar un segundo parseador para traducir la misma sentencia descrita anteriormente a una aceptada por la librería ABC4J, la cual permite generar pentagramas.

**ABC4J** ABC4J es una librería para el lenguaje de programación Java que permite generar pentagramas en forma gráfica a partir de un string, el cual debe ser formateado de tal forma que coincida con la notación aceptada por la librería.

**Tarjeta de Sonido** Una vez que el parseador interno traduce una sentencia a frecuencia y duración, esta debe ser interpretada y luego enviada a la tarjeta de sonido como un buffer de salida de bytes. De esta forma, es comprobado que DSL Music no posee sonidos por defecto, sino que los genera convirtiendo una frecuencia a arreglos de bytes.

### 5.1.3. Salidas

#### Música

Mediante una función en Java, los arreglos de bytes que representan una frecuencia medida en Hertz, son expulsados como sonidos, los cuales no sufren variaciones al momento de ejecutar la aplicación en distintos dispositivos, por lo cual, el resultado siempre será el mismo.

#### Pentagrama

Mediante Java y ABC4J, se crea un pentagrama en forma gráfica que representa de forma idéntica la melodía que se ha compuesto.

## 5.2. Modelamiento diagrama flujo de datos

Un diagrama de flujo de datos (DFD sus siglas en español e inglés) es una representación gráfica del flujo de datos a través de un sistema informático. Un diagrama de flujo de datos

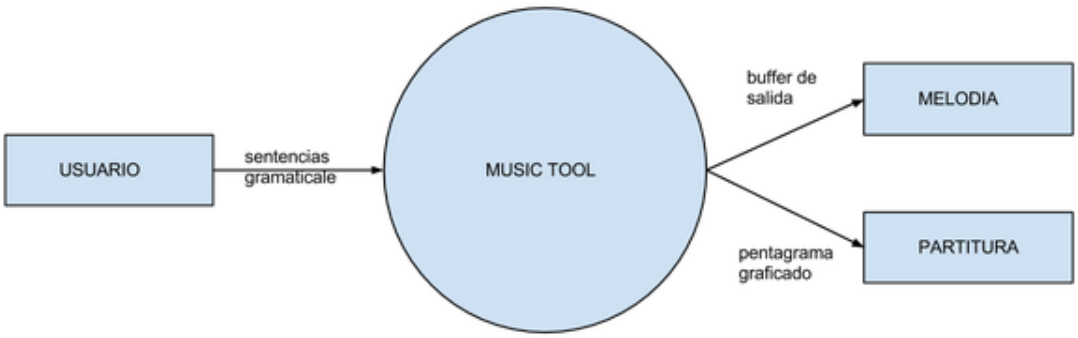
también se puede utilizar para la visualización de procesamiento de datos (diseño estructurado). Es una práctica común para un diseñador dibujar un contexto a nivel de DFD que primero muestra la interacción entre el sistema y las entidades externas.

Los diagramas de flujo de datos pueden ser usados para proporcionar al usuario final una idea física de cómo resultarán los datos a última instancia, y cómo tienen un efecto sobre la estructura de todo el sistema. La manera en que cualquier sistema es desarrollado, puede determinarse a través de un diagrama de flujo de datos.

Los diagramas pueden ser de tres tipo, diagrama de contexto (nivel 0), diagrama de nivel superior (nivel 1) y diagrama de detalle (nivel 2). Para cada uno de estos diagramas, el nivel de abstracción se hace cada vez menor, posibilitando el entendimiento de los desarrolladores al momento de implementar cada módulo del sistema.

Los elementos que componen un diagrama de flujo de datos son cuatro, de los cuales tres son los ocupados para representar DSL Music: Proceso (se representa mediante un círculo), Entidad Externa (corresponde a un rectángulo) y el Flujo de Datos (representado con una flecha).

### 5.2.1. Diagrama de contexto (Nivel0)




**Entidades externas**

- Usuario: es aquél que mediante eventos genera sentencias gramaticales que luego envía a procesarlas.
- Melodía: corresponde a una de las salidas de la aplicación, se muestra en forma de sonido.
- Partitura: corresponde a la representación gráfica de una composición hecha por el usuario.

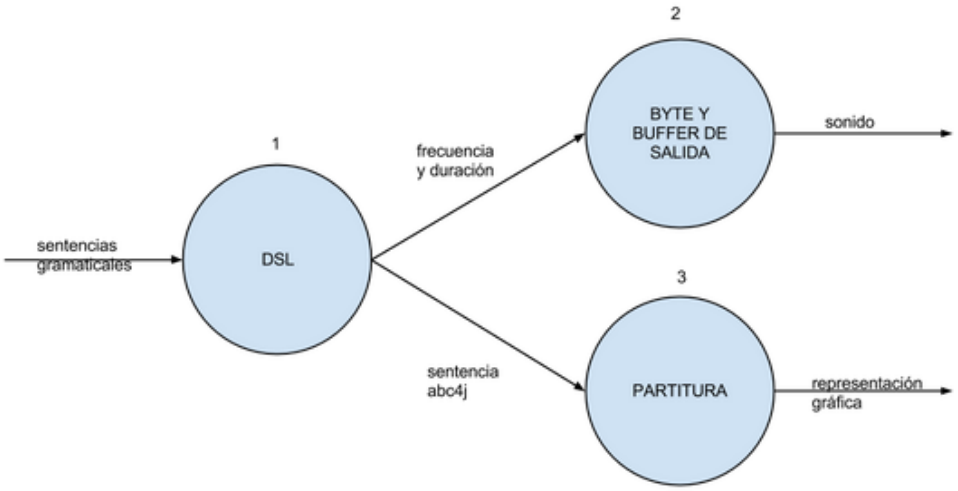
**Procesos**

- DSLMusic: corresponde al proceso principal de la aplicación en donde las entradas de usuario son procesadas para poder generar una salida.

**Flujos**

- Sentencias gramaticales: corresponde a las entradas de usuario, ya sea por teclado o por archivo.
- Buffer de salida: corresponde a los arreglos de bytes enviados a la tarjeta de sonido para luego emitir el sonido.
- Pentagrama Graficado: corresponde a la composición generada por el usuario representada en forma gráfica, la cual es insertada en un Jpanel.

### 5.2.2. Diagrama de Nivel Superior (Nivel 1)




**Procesos**

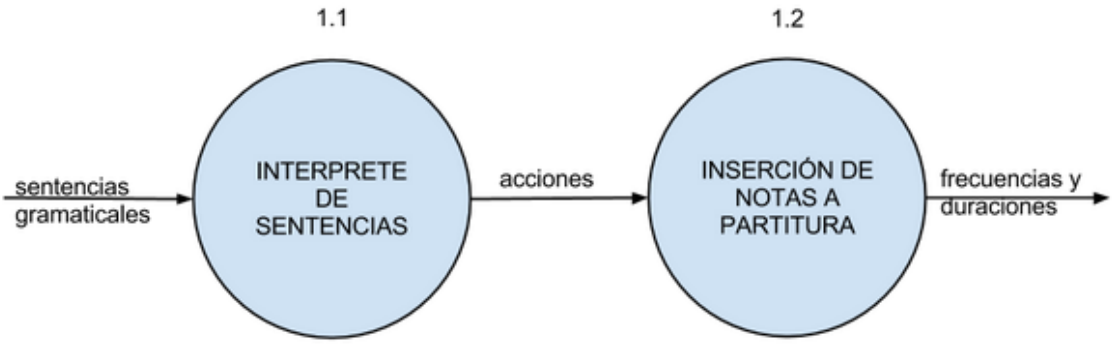
- DSL: proceso encargado de reconocer e interpretar sentencias gramaticales para luego generar acciones necesarias para la exportación final.
- BYTE Y BUFFER DE SALIDA: proceso encargado de traducir una frecuencia y duración (correspondiente a una nota) en arreglo de bytes para enviarlos al buffer de salida de la tarjeta de sonido del equipo donde sea utilizada la aplicación.
- PARTITURA: proceso encargado de traducir una sentencia en formato ABC4J a una representación gráfica.

**Flujos**

- Sentencias gramaticales: corresponde a las entradas de usuario, ya sea por teclado o por archivo.
- Frecuencia y duración: con ANTLR, el proceso DSL como salida envía una frecuencia medida en Hertz y una duración en milisegundos que luego será tratada por el proceso de byte y buffer de salida.

- Sentencia ABC4J: el proceso DSL una vez que reconoce las sentencias gramaticales ingresadas por el usuario, mediante acciones se hace el llamado a un parser interno para que pueda reconocer y traducir esta sentencia.
- Sonido: cada sentencia gramatical bien escrita es traducida en sonido para luego con un conjunto de ellos generar una melodía.
- Representación Gráfica: corresponde a la composición generada por el usuario representada en forma gráfica, la cual es insertada en un Jpanel.

### 5.2.3. Diagrama de detalle DSL (nivel 2)



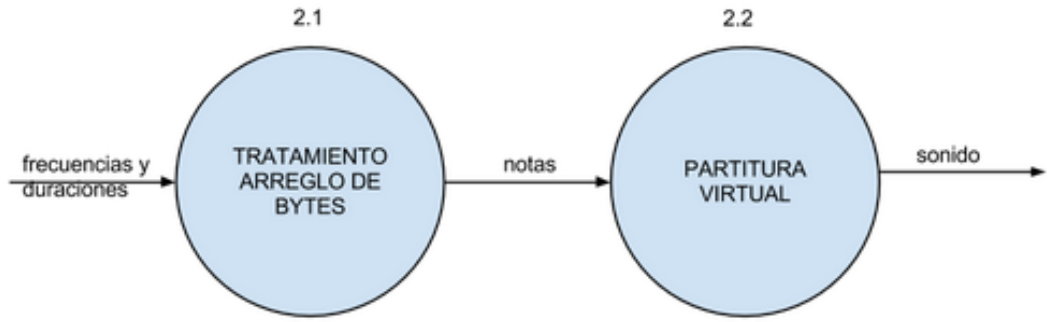

**Procesos**

- INTÉRPRETE DE SENTENCIAS: por cada sentencia gramatical escrita, el proceso de interpretación se encarga de verificar que las entradas de usuario estén correctas para luego generar acciones..
- INSERCIÓN DE NOTAS A PARTITURA: las acciones conllevan a generar código Java que permite crear una partitura virtual con cada sentencia según corresponda.

**Flujos**

- Sentencias gramaticales: corresponde a las entradas de usuario, ya sea por teclado o por archivo.
- Acciones: este flujo corresponde a la ejecución de código Java ligado al reconocimiento de una sentencia gramatical bien escrita.
- Frecuencia y duración: las notas ingresadas a partitura son expulsadas en forma de frecuencia y duración a un proceso de traducción a arreglo de bytes.

### 5.2.4. Diagrama de detalle: byte y buffer de salida(Nivel 2)



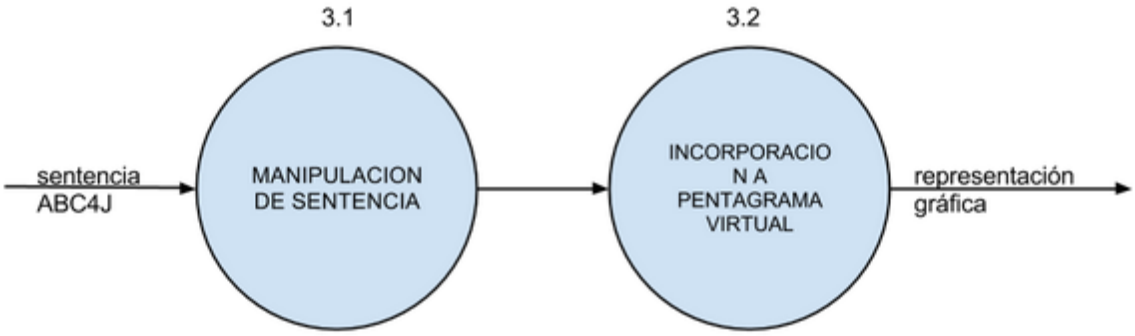

**Procesos**

- TRATAMIENTO ARREGLO DE BYTE: una vez recibidas las listas de frecuencias y duraciones de una composición, estas son transformadas en arreglos de bytes para luego ser enviadas al buffer de salida de la tarjeta de sonido del equipo donde será ejecutada la aplicación.
- PARTITURA VIRTUAL: cada arreglo de byte es añadido a una partitura virtual de manera tal que un conjunto de sonidos, sean reproducidos como una melodía.

**Flujos**

- Frecuencias y duraciones: las notas ingresadas a partitura son expulsadas en forma de frecuencia y duración a un proceso de traducción a arreglo de byte.
- Notas: a cada arreglo de bytes se le hace corresponder una nota musical en particular la cual será enviada al proceso que genera la partitura virtual.
- Sonido: el proceso de partitura virtual genera este flujo de salida que corresponde a la composición del usuario en forma de sonido.

### 5.2.5. Diagrama de detalle: pentagrama(Nivel 2)




**Procesos**

- MANIPULACIÓN DE SENTENCIA: el proceso requiere como entrada una sentencia en formato ABC4J, la cual es formateada para añadirla a la librería.
- INCORPORACIÓN A PENTAGRAMA VIRTUAL: este proceso se encarga de añadir cada figura musical a un pentagrama virtual el cual luego es mostrado gráficamente como salida.

**Flujos**

- Sentencia ABC4J: corresponde a una cadena de caracteres que contiene símbolos especiales que serán transformados en figuras musicales de forma gráfica.
- Representación Gráfica: corresponde a la composición generada por el usuario representada en forma gráfica, la cual es insertada en un Jpanel.

# Capítulo 6

# Lecciones Aprendidas

## 6.1. ABC4J: Dimitry Castex y Marcelo Valdivia

La generación de un pentagrama fue caótico, pues las opciones para cumplir este requerimiento funcional de la aplicación eran demasiadas, por un lado, generando imágenes y renderizándolas en pantalla o tal vez generando código LaTeX que pudiese representar una partitura. Estas opciones eran infactibles considerando el tiempo restante para finalizar el curso.

Para solucionar esta situación optamos por trabajar con una librería encontrada de forma inesperada llamada ABC4J, la cual mediante sentencias gramaticales en un formato específico crea un pentagrama con figuras musicales y las muestra dentro de un JPanel.

El único inconveniente de esta librería fue su poca documentación, lo cual causó problemas al momento de generar todas las figuras musicales y silencios de forma correcta. La forma de solucionar ésto fue usando la notación ABC creada en Estados Unidos para aspectos musicales, la cual fue compatible casi completamente con la librería.

## 6.2. ANTLR: Marcelo Valdivia

Si bien Dimitry Castex ya se manejaba en el uso de ANTLR, para mi fue una experiencia un tanto complicada, pues asociar la parte teórica del curso a la parte práctica me costó más de la cuenta, generando así complicaciones al momento de crear la gramática de nuestro proyecto.

Esta situación superada gracias a la documentación para ANTLR v3 en el sitio oficial.

## 6.3. Problemas con versiones de ANTLR Works: Dimitry Castex

ANTLR Works generó problemas en mi computador pero no así en el de mi compañero, con lo que tuve que probar con varias versiones pero nada dio resultado.

En primera instancia, la solución fue trabajar solo en el computador de mi compañero. La opción final, instalar un plugin de ANTLR en Eclipse y todo funcionó correctamente.

## 6.4. Anexos

### 6.4.1. Theremin virtual de Jorge Rubira

```
import java.awt.Color;
import java.awt.Graphics;
import java.awt.event.*;
import javax.sound.sampled.*;
import javax.swing.JFrame;
public class Theremin extends JFrame implements Runnable{
private int frecuencia=440;  // Frecuencia Hz del sonido
private double volumen=0;    // Volumen del sonido [0...1]
private boolean bSeguir=true;  //Finalizar la iteración del hilo
private byte onda[]=new byte[ 100 ];   //Forma de la onda
private int escala=2;  //Escala de visualización
private SourceDataLine line=null;   //Salida del audio
private boolean vibracion=false;   //Activa la vibración
private boolean autoajuste=false;  //Activa el autoajuste - entonar la nota cuando está
private double escalaDiatonica[]=new double[12*6];
public Theremin(){
//Inicialmente genera una onda sinusoidal
for (int n=0;n < 100;n++){
onda[n]=(byte)(Math.sin(n*Math.PI*2/100)*127);
}
//Calcula las lineas a visualizar de la escala diatonica
calcularEscalaDiatonica();
//Atributos de la ventana
setVisible(true);
setSize(800,200);
setResizable(false);
setTitle("Theremin Virtual. by Jorge Rubira");
//Eventos del mouse
this.addMouseListener(new MouseListener() {
public void mousePressed(MouseEvent e) {
//Al pulsar el boton
if ((e.getButton() & MouseEvent.BUTTON2) > 0){
int n=e.getX()*100/getWidth(); //Obtiene la posición del vector
int t=-(e.getY()-100)*135/80; //Obtiene la amplitud clickeada.
int factor=1; //Determina si tiene que subir o bajar el valor según si clickea por arrib
if (onda[n]>t){
factor=-1;
}
//Para que no dé chasquidos modifica la posición seleccionada y los 20 más cercanos (cor
for (int z=0;z < 10;z++){
int z2=(n+z)%onda.length;
onda[z2]=(byte)limite(onda[z2]+(10-z)*factor*2,-127,127);
```

```
if (z>0){
z2=n-z;
if (z2 < 0){
z2+=onda.length;
}
onda[z2]=(byte)limite(onda[z2]+(10-z)*factor*2,-127,127);
}
}
}else if ((e.getButton() & MouseEvent.BUTTON1) > 0){
//Activa la vibración
vibracion=true;
if (e.getClickCount()==2){
//El doble click activa/desactiva el autoajuste de la nota
autoajuste=!autoajuste;
}
}
repaint();
}
public void mouseReleased(MouseEvent e) {
if ((e.getButton() & MouseEvent.BUTTON1) > 0){
//Desactiva la vibración
vibracion=false;
}
repaint();
}
public void mouseExited(MouseEvent e) {}
public void mouseEntered(MouseEvent e) {}
public void mouseClicked(MouseEvent e) {}
});
this.addMouseMotionListener(new MouseMotionListener() {
public void mouseMoved(MouseEvent e) {
volumen=limite((200-e.getY())-75,0,50)/50.0;
frecuencia=autoajuste(e.getX())*escala+100;
}
public void mouseDragged(MouseEvent e) {mouseMoved(e);}
});
this.addWindowListener(new WindowListener() {
public void windowClosing(WindowEvent e) {
//Finaliza la aplicación cerrando el dispositivo del audio.
bSeguir=false;
        line.drain();
        line.close();
        System.exit(0);
}
public void windowOpened(WindowEvent e) {}
```

```
public void windowIconified(WindowEvent e) {}
public void windowDeiconified(WindowEvent e) {}
public void windowDeactivated(WindowEvent e) {}
public void windowClosed(WindowEvent e) {}
public void windowActivated(WindowEvent e) {}
});
try{
//Abre el canal del audio
AudioFormat af = new AudioFormat(22000, 8, 1, true, true);
     line = AudioSystem.getSourceDataLine(af);
line.open(af, 22000);
line.start();
new Thread(this).start();
}catch(Exception e){
e.printStackTrace();
}
}
public void update(Graphics g){
paint(g);
}
public void paint(Graphics g){
//Limpia la ventana
g.clearRect(0, 0, getWidth(), 200);
//Pinta la señal sinusoidal (o la editada)
g.setColor(Color.blue);
for (int n=0;n < getWidth ();n++){
g.drawLine(n, 100-onda[n*100/getWidth()]*80/135, n, 100-onda[n*100/getWidth()]*80/135);
}
//Pinta las franjas rojas límites de variación de amplitud
g.setColor(Color.red);
g.drawLine(0, 75, getWidth(), 75);
g.drawLine(0, 125, getWidth(), 125);
//Pinta las notas de la escala diatónica
g.setColor(Color.black);
for (int not=0;not < escalaDiatonica.length;not++){
int nX=(int)escalaDiatonica[not];
g.drawLine(nX, 0, nX, 200);
g.drawString(""+(char)(not%12+(byte)'A'), nX, 190);
}
//Pinta mensajes de estados (vibración y autoajuste)
if (vibracion){
g.drawString("Vibracion", 700, 40);
}
if (autoajuste){
g.drawString("Autoajuste", 700, 60);
```

```
}
}
public int autoajuste(int x){
if (!autoajuste) return x;
//Si el ratón está cerca de la nota, emite la nota exacta
for (int not=0;not < escalaDiatonica.length;not++){
if (x>escalaDiatonica[not]-11 && x < escalaDiatonica[not]+11){
return (int)escalaDiatonica[not];
}
}
return x;
}

public void calcularEscalaDiatonica(){
//Calcula la escala diatónica
double ed[]={0, 2, 3, 5, 7, 8, 10};
for (int not=0;not < ed.length;not++){
ed[not]=Math.pow(2, ed[not]/12);
}
for (int oct=0;oct < 6;oct++){
for (int not=0;not < ed.length;not++){
int nX=((int)(110*Math.pow(2, oct)*ed[not])-100)/escala;
escalaDiatonica[oct*12+not]=nX;
}
}
}

@Override
public void run() {
double ang=0;  //Angulo de la señal de audio en cada momento
byte b[]=new byte[ 100 ];  //Tamaño máximo del buffer
//Al utilizar 22000 Hz inicialmente se muestrará con 22 muestras por milisegundo.
int bufferAjuste=22;
double ang2=0;
//Ejecuta hasta finalizar la aplicación
while(bSeguir){
for (int z=0;z < bufferAjuste;z++){
//Calculo del angulo según la frecuencia seleccionada con el ratón
ang+=Math.PI*2*frecuencia/22000;
while(ang>Math.PI*2){
ang-=Math.PI*2;
}
//Obtiene el valor de la onda
b[z]=(byte)(onda[(int)((ang*100/(Math.PI*2)))%onda.length]*volumen);
//Si hay vibración multiplica la señal por otra señal sinusoidal con rango [0.5-1]
```

```
if (vibracion){
b[z]=(byte)limite(b[z]*(Math.cos(ang2)/2+1),-127,127);
ang2+=0.0025;
if (ang2>Math.PI*2){
ang2-=Math.PI*2;
}
}else{
//Si no hay vibración resetea la señal sinusoidal
ang2=0;
}
}
//Vuelca los valores al buffer
line.write(b, 0, bufferAjuste);
//Espera 1 milisegundo
     try{
     Thread.sleep(1);
     }catch(Exception e){}

     //Variación de las muestras a volcar por milisegundo
     int nDes=0;
     if (line.getBufferSize()-line.available() < 80){
     nDes=1;
     }
     if (line.getBufferSize()-line.available()>3000){
     nDes=-1;
     }
     bufferAjuste+=nDes;
     if (bufferAjuste>100){
     bufferAjuste=100;
     }
}
}

//Funciones de delimitación de rangos
public int limite(int valor, int min, int max){
return Math.min(Math.max(valor,min),max);
}
public int limite(double valor, int min, int max){
return Math.min(Math.max((int)valor,min),max);
}
//Inicio de la aplicación
public static void main(String arg[]){
new Theremin();
}
}
```

### 6.4.2. DSLMusic

**Nota**

```
package musica;

public class Nota
{

double frecuencia = 440;
double duracion = 22000;
//arreglo que crea la onda a reproducir
byte b[];

public Nota(double frec,double dur)
{
//para calcular la frecuencia, necesito la nota y la octava
frecuencia = frec;
duracion = dur*22;
b=new byte[(int)duracion]; //Buffer

for (int n=0;n < b .length;n++)
{
//22000 es la frecuencia de muestreo del sonido
//127 es la amplitud máxima de un byte [-127,127]. Obviamos el 128 para hacerlo más fáci
b[n]=(byte)(Math.sin(frecuencia*n*Math.PI*2/22000)*127);
}
}
}
```

**Partitura**

```
package musica;

import java.util.ArrayList;
import java.util.Iterator;

import javax.sound.sampled.AudioFormat;
import javax.sound.sampled.AudioSystem;
import javax.sound.sampled.SourceDataLine;

//El contenedor de notas será el que reproduce todas las notas, por eso extende de threa
public class Partitura extends Thread
{
//arreglo ue contendrá las notas
private ArrayList<Nota> arregloNotas;
//iterador para recorrer arreglo
```

```
private Iterator<Nota> it;
//numero para ms
double ms=1000;
//linea de audio
SourceDataLine line;


//partitura para gramaticas de figuras
//se toma en cuenta el valor milisec solo cuando hay gramatica de figuras, en el caso
//de las gramaticas fijas no se toma en cuenta, pero se agrega un valor pro defecto para
public Partitura()
{
//inicio el arreglo de notas
arregloNotas = new ArrayList<Nota>();
}
//para setear los milisegundos luego de crear el objeto
public void setMilisegundos(double milisec)
{
ms = milisec;
}

//=== PARTITURA PARA GRAMÁTICAS DE FIGURAS ===
public void addNota(String nota, String octava, String duracion)
{
addNota(TraductorInterno.StringToFrecuency(nota, octava),TraductorInterno.StringToDurati
}

public void addSilencio(String duracion)
{
addSilencio(TraductorInterno.StringToDuration(duracion, ms));
}

//=== PARTITURA PARA GRAMÁTICAS DE FRECUENCIAS ===
//agrega una nota a la partitura
public void addNota(double frecuencia,double duracion)
{
Nota n = new Nota(frecuencia,duracion);
arregloNotas.add(n);
}

//agrega un silencio a la partitura
public void addSilencio(double duracion)
{
Nota n = new Nota(0,duracion);
arregloNotas.add(n);
```

```
}

public void run()
{
//linea para lanzar audio
line = null;
//Abre el dispositivo de salida
try
{
//asigno el audioformat
AudioFormat af = new AudioFormat(22000, 8, 1, true, true);
//asigno al sistema de audio el audioformat
line = AudioSystem.getSourceDataLine(af);
//asigno el iterador
it = arregloNotas.iterator();
//abro la linea de audio
line.open(af, 22000);
//empiezo a lanzar las muestras
line.start();
//mientras hayan notas
while(it.hasNext())
{
//recorro el arreglo para reproducir notas
Nota notaActual = it.next();
//Vuelca la señal
line.write(notaActual.b, 0, notaActual.b.length);
}
//Finaliza a que se emita todo el sonido
line.drain();
//Cierra la linea
line.close();
}
catch(Exception e)
{
e.printStackTrace();
}
}

public void parar()
{
line.close();
this.interrupt();
}

}
```

### Pentagrama

```
package musica;

import gui.PentagramaGrafico;

import java.util.ArrayList;
import java.util.Iterator;

public class Pentagrama
{
int contador=0;
//arreglo que contendrá las notas
//importante agregar un espacio despues del \n
String pentagramaTotal="X:0\nL:1/32\n ";

public void addNota(String nota,String octava,String duracion)
{
pentagramaTotal+=TraductorInterno.NotaToABC(nota, octava, duracion)+" ";
}

public void addSilencio(String duracion)
{
pentagramaTotal+=TraductorInterno.SilencioToABC(duracion)+" ";
}

public void start()
{
PentagramaGrafico p = new PentagramaGrafico(pentagramaTotal);
p.setVisible(true);
}

}
```

### Traductor Interno

```
package musica;

import java.util.Locale;

//formula: frecuencia = 440 *  Math.pow(1.059463, (octava-4)*12+(nota-10));

public class TraductorInterno
```

```
{

public static double StringToFrecuency(String nota,String octava)
{
//parseo a double los valores que recibo
double n=0;
double o;

switch(nota)
{
case   "DO": n=1;break;
case  "DOS": n=2;break;
case   "RE": n=3;break;
case  "RES": n=4;break;
case   "MI": n=5;break;
case   "FA": n=6;break;
case  "FAS": n=7;break;
case  "SOL": n=8;break;
case "SOLS": n=9;break;
case   "LA": n=10;break;
case  "LAS": n=11;break;
case   "SI": n=12;break;
}

o = Double.parseDouble(octava);

//los ingreso a la formula y lo retorno
return(440 *  Math.pow(1.059463, (o-4)*12+(n-10)));
}

public static double StringToDuration(String duracion,double milisegundos)
{
int division=1;
//duracion en milisegundos
switch(duracion)
{
case     "REDONDA":division=1;break;
case      "BLANCA":division=2;break;
case       "NEGRA":division=4;break;
case     "CORCHEA":division=8;break;
case "SEMICORCHEA":division=16;break;
case        "FUSA":division=32;break;
}
//divido los milisegundos
return(milisegundos/division);
```

```
}

public static String NotaToABC(String nota,String octava,String duracion)
{
String ABC="";
//primero, dependiendo de la nota elegiremos la letra en mayuscula
switch(nota)
{
case   "DO": ABC+="C";break;
case  "DOS": ABC+="^C";break;
case   "RE": ABC+="D";break;
case  "RES": ABC+="^D";break;
case   "MI": ABC+="E";break;
case   "FA": ABC+="F";break;
case  "FAS": ABC+="^F";break;
case  "SOL": ABC+="G";break;
case "SOLS": ABC+="^G";break;
case   "LA": ABC+="A";break;
case  "LAS": ABC+="^A";break;
case   "SI": ABC+="B";break;
}

//luego, dependiendo de la octava, la letra se pasará a minuscula o bien, se le agregará
int o = Integer.parseInt(octava);



/* octava < 2 = mayuscula ,
 * octava   2 = mayuscula
 * octava   3 = minuscula
 * octava > 3 = minuscula '
 */

//en estos dos bloques, ya se está en mayuscula
if(o < 2)
{
//ajustes para agregar la cantidad deseada de comas
int a=(o-2)*-1;

for(int i=0;i<a;i++)
{
ABC = ABC.concat(",");
}
```

```
}
else
//no se necesita o==2, ya que es por defecto una letra mayuscula, que ya está puesta
//acá se necesita pasar a minusculas
if(o == 3)
{
ABC = ABC.toLowerCase(Locale.ENGLISH);
}
else
if(o > 3) //además de pasar a minusculas, se necesita agregar '
{
ABC.toLowerCase(Locale.ENGLISH);

for(int i=0;i<o-3;i++)
{
ABC = ABC.concat("'");
}
}

String d="";

switch(duracion)
{
case     "REDONDA":d="8" ;break;
case      "BLANCA":d="4" ;break;
case       "NEGRA":d="2" ;break;
case     "CORCHEA":d=""  ;break;
case "SEMICORCHEA":d="/2";break;
case        "FUSA":d="/4";break;

}

//y lo concatenamos
ABC = ABC.concat(d);
// para finalmente retornar la nota
return ABC;

}

public static String SilencioToABC(String duracion)
{
String ABC="z";
String d="";

switch(duracion)
```

```
{
case     "REDONDA":d="8";break;
case      "BLANCA":d="4";break;
case       "NEGRA":d="2";break;
case     "CORCHEA":d="";break;
case "SEMICORCHEA":d="/2";break;
case        "FUSA":d="/4";break;
}

ABC = ABC.concat(d);

return ABC;
}
}
```

### 6.4.3. Gramática

```
grammar DSLMusic;

@parser::header
{
 package dsl;

 import java.util.ArrayList;
 //clases propias
 import musica.Partitura;
 import musica.Pentagrama;
}

@lexer::header
{
 package dsl;

 import java.util.ArrayList;
}

@lexer::members
{
  //arreglo para guardar los errores
  public ArrayList<String> erroneos = new ArrayList<String>();

  public void emitErrorMessage(String msg)
  {
    erroneos.add("Error:"+msg);
  }
```

```
}

@parser::members
{
  //arreglo para guardar los errores
  public ArrayList<String> erroneos = new ArrayList<String>();
  //se declara una partitura, para crearla dependiendo si es la gramática grupo_fijo o g
  public Partitura partitura = new Partitura();
  //se crea un pentagrama, que se ocupará solo para grupo_partitura
  public Pentagrama pentagrama = new Pentagrama();
  //variable que indica si es gramatica de figuras y gramatica fija
  public boolean figuras;
  //para el manejo de errores
  public void emitErrorMessage(String msg)
  {
    erroneos.add("Error:"+msg);
  }
}


//union de gramaticas
start : grupo_partitura|grupo_fijo;

//REGLAS
//grupo partitura
grupo_partitura: def_ms (partitura_nota|partitura_silencio)+ 'END' {figuras=true;};

def_ms  : 'UNIDAD=' tiempo';'{partitura.setMilisegundos($tiempo.value);};

tiempo returns[double value]  : NUMERO octava* unidadtiempo {$value = Double.parseDouble

unidadtiempo  : UNIDADTIEMPO;

partitura_nota:'NOTA(' nota octava ',' figuras ');' {partitura.addNota($nota.text,$octav

partitura_silencio:'SILENCIO('figuras');' {partitura.addSilencio($figuras.text);pentagra

octava  :  NUMERO;

figuras : FIGURAS;

nota: DO | DOS | RE | RES | MI | FA | FAS | SOL | SOLS | LA | LAS | SI;

//grupo fijo
grupo_fijo: (fijo_sonido|fijo_silencio)+ 'END' {figuras=false;};
```

```
fijo_sonido : 'SONIDO(' frecuencia ',' tiempo ');'{partitura.addNota($frecuencia.frec_va

fijo_silencio   : 'SILENCIO('tiempo');'{partitura.addSilencio($tiempo.value);};

frecuencia returns[double frec_value]:  NUMERO unidadfrec{$frec_value = Double.parseDoub

unidadfrec:  UNIDADFREC;

//TERMINALES
//grupo partitura
DO:'DO';

DOS:'DOS';

RE:'RE';

RES:'RES';

MI: 'MI';

FA: 'FA';

FAS: 'FAS';

SOL: 'SOL';

SOLS: 'SOLS';

LA: 'LA';

LAS: 'LAS';

SI: 'SI';

NUMERO  :'0'..'9'+;

FIGURAS:'REDONDA' | 'BLANCA' | 'NEGRA' | 'CORCHEA' | 'SEMICORCHEA' | 'FUSA';

//grupo fijo
UNIDADFREC : 'HZ';

UNIDADTIEMPO: 'MS';

//salto de línea
```

```
WS  :   ( ' '
        | '\t'
        | '\r'
        | '\n'
        ) {$channel=HIDDEN;};
```