UNIVERSIDAD CARLOS III DE MADRID

ESCUELA POLITÉCNICA SUPERIOR

SISTEMA INTELIGENTE PARA COMPOSICIÓN ARMÓNICA MUSICAL

PROYECTO FIN DE CARRERA
INGENIERÍA TÉCNICA DE TELECOMUNICACIÓN:
SONIDO E IMAGEN

Autora: Celia Clemente Castillo
Tutor: Julio Villena Román

Junio 2010

Título: Sistema inteligente para composición armónica musical

Autor: Celia Clemente Castillo

Tutor: Julio Villena Román

EL TRIBUNAL

Presidente:
Ricardo Romeral Ortega

Secretario:
Iria Estévez Ayres

Vocal:
Aitor Mendaza Ormaza

Realizado el acto de defensa del Proyecto Fin de Carrera el día 9 de Junio de 2010 en Leganés, en la Escuela Politécnica Superior de la Universidad Carlos III de Madrid, acuerda otorgarle la CALIFICACIÓN de:

Fdo: Presidente Fdo: Secretario Fdo: Vocal

A la memoria de Felipa, Gabriel e Hilario

Agradecimientos

Quiero agradecer este proyecto y todo el conocimiento y esfuerzo llevado a cabo para llegar hasta él a mi familia por el apoyo y la ilusión aportada a lo largo de todos estos años, a Honza, a los Sonimágicos por haber hecho este periodo algo inolvidable, al grupo de la biblioteca por tantas horas y vivencias invertidas, a todos los que han pasado por mi vida por la huella que han dejado en ella, a Luis, a Javi, a Clara, a Lucia, a Sara, a Isa M. Rosa, a Jess, a Pedrosa, a todos aquellos que me han contagiado la pasión por la música, a Julio por toda la paciencia que ha tenido conmigo, a los amigos de TID por ese apoyo moral dado durante la realización de este proyecto, y a todo el resto de mis amigos por ser eso, amigos.

Resumen

El presente proyecto consiste en el diseño e implementación de un sistema experto para composición armónica musical coral en diferentes estilos. Para obtener un resultado habrá que indicar el estilo, representado por un conjunto de reglas, y la melodía a armonizar que se corresponderá con la voz más aguda del coro. El módulo principal hará un análisis de la entrada para determinar qué notas pueden usarse para la armonización, sin poner otra condición que la nota a armonizar, y será el módulo de conocimiento, implementado mediante un sistema experto, el que tomará las decisiones de análisis armónico y notas usadas para ello.

ÍNDICE

1. INTRODUCCIÓN 1
1.1. Motivación del proyecto 1
1.2. Objetivos del proyecto 1
1.3. Estructura de la memoria 2
2. ESTADO DEL ARTE 4
2.1. Composición musical y matemáticas. 4
2.2. Sistemas expertos 7
2.2.1. Herramientas y Lenguajes 9
2.3. Música e Inteligencia Artificial. 16
2.3.1. Arca Musarithmica 18
2.3.2. Musikalishcher Würfelspier 20
2.3.3. Componium 22
2.3.4. Illiac Suit 23
2.4. Sistemas actuales de composición por ordenador 24
2.4.1. Proyecto EMI de David Cope 25
2.4.2. Designing Music 25
2.4.3. SICOM 27
2.4.4. Finale 28
3. ARMONIZADOR 30
3.1. Definiciones 30
3.2. Diseño del sistema 41
3.2.1. Estructura del sistema 42
3.2.2. Gestión de información 44
3.2.3. Clases Java 46
4 IMPLEMENTACIÓN DE REGLAS 67
4.1. CoralClasica 67
4.2. Aleatorio 82
5. EVALUACIÓN 84
5.1. Entrada Do Mayor compás 6/8 84
5.2. Entrada Sib Mayor compás 3/4 87
5.3. Entrada Himno de la Alegría 90
5.4. Entrada Bacarola 92

6. CONCLUSIONES Y TRABAJOS FUTUROS....96
6.1. Conclusiones....96
6.2. Trabajos futuros....98
Apéndice A: Presupuesto....101
BIBLIOGRAFÍA....103

ÍNDICE DE FIGURAS

Figura 1. Regla en CLIPS 10
Figura 2. Regla Jess 11
Figura 3. Regla Drools 12
Figura 5. Composición de archivo de reglas 13
Figura 4. Vista de alto nivel de un Motor de Reglas 13
Figura 6. Estructura de una regla 14
Figura 7. Estructura de datos de JMusic 15
Figura 8. Diseño original del arca musarithmica 19
Figura 9. Arca Musarithmica 20
Figura 10. Tablas del juego de dados de Mozart 21
Figura 11. Fotografía de un componium 23
Figura 12. Fotografía del ordenador ILLIAC I 24
Figura 13. Estructura de DM-D 26
Figura 14. Jerarquía del análisis llevado a cabo por SICOM 27
Figura 15. Relación horizontal entre estructuras 28
Figura 16. Captura de Finale de composición de percusión 29
Figura 17. Captura de pantalla de Finale 29
Figura 18. Acorde de tríada de Re Mayor. 30
Figura 19. Acorde de cuatríada de Do Séptima. 31
Figura 20. Acorde de Re Mayor. 31
Figura 21. Acorde de Re menor. 31
Figura 22. Secuencia de acordes 31
Figura 23. Escala en Sol mayor 32
Figura 24. Escala de Mi menor 32
Figura 25. Tipos de movimientos entre dos voces 32
Figura 26. Tesituras de las voces de los coros. 33
Figura 27. Representación de todas las claves musicales 33
Figura 28. Compás de 4/4 34
Figura 29. Correspondencia entre figuras y denominadores 34
Figura 30. Correspondencia entre figuras y silencios 34

Figura 31. Correspondencia de valores relativos....35
Figura 32. Enlaces posibles entre los grados de las tonalidades 39
Figura 33. Notas que se armonizan en un compás de 6/8....40
Figura 34. Diagrama de bloques del sistema....42
Figura 35. Entrada genérica al sistema 45
Figura 36. Salida del sistema tras las decisiones 45
Figura 37. Diagrama de clases Nota, NotaFinal y NotaPosible 47
Figura 38. Diagrama de clases Armonizador y Compositor....47
Figura 39. Constructor a partir de todos los atributos 50
Figura 40. Constructor a partir de otro objeto Nota....50
Figura 41. Método toString() de la clase Nota 51
Figura 42. Clase NotaFinal que extiende de Nota 51
Figura 43. Clase NotaPosible que extiende de Nota 52
Figura 44. Fragmentos de código con la creación de la StatefullKnowledgeSession 53
Figura 45. Inicialización del entorno Drools 55
Figura 46. Análisis de acordes para armonizar la melodía 56
Figura 47. Escala de Do Mayor 57
Figura 48. Grados de Do Mayor 57
Figura 49. Escala de Do menor 57
Figura 50. Grados de Do menor 57
Figura 51. Inserción de notas posibles en la sesión 59
Figura 52. Obtención de los datos del archivo MIDI 60
Figura 53. Cálculo según el compás de que notas hay que armonizar 61
Figura 54. Cálculo de los valores de las variables de la clase Nota para notas MIDI....62
Figura 55. Adaptación del valor de la duración de las notas....63
Figura 56. Decisión de que notas se armonizan....63
Figura 57. Creación del objeto de la clase Composición 64
Figura 58. Ordenación del resultado....65
Figura 59. Escritura de archivo MIDI a partir de la información devuelta por Compositor .66
Figura 60. Conversión de Nota a Note 66
Figura 61. Acotación de las voces contralto, tenor y bajo....68
Figura 62. Regla que elimina los valores posibles cuando existe un valor final....69

Figura 63. Regla que cuando solo existe una nota posible la pasa a resuelta 70
Figura 64. Regla que elimina los acordes que no aparece en las otras voces 70
Figura 65. Regla que fuerza a que todas las notas aparezcan en cada posición 71
Figura 66. Reglas para controlar la distancia permitida entre voces 72
Figura 67. Reglas para que las voces no se crucen 73
Figura 68. Reglas quintas consecutivas entre soprano y contralto 74
Figura 69. Ejemplo de regla de quinta consecutiva. Contralto-tenor 76
Figura 70. Ejemplo de regla de octavas consecutivas. Soprano-contralto 77
Figura 71. Regla para que el primer acorde sea de grado I 77
Figura 72. Regla para la eliminación del acorde VII 78
Figura 73. Reglas de enlaces con el grado I 79
Figura 74. Reglas de enlaces condicionados por el grado II 81
Figura 75. Regla que elimina el resto de acordes cuando se elige uno 81
Figura 76. Regla de elección aleatoria 82
Figura 77. Reglas del archivo aleatorio.drl 83
Figura 78. Información de entrada del sistema CM 84
Figura 79. Partitura en Do mayor introducida al sistema 85
Figura 80. Análisis resultante del sistema. Do mayor 85
Figura 81. Salida del sistema Do mayor reglas coral clásica 86
Figura 82. Salida del sistema Do mayor reglas aleatorias 87
Figura 83. Información de entrada al sistema BbM 87
Figura 84. Partitura en Sib mayor introducida al sistema 88
Figura 85. Análisis resultante del sistema. Sib mayor 88
Figura 86. Salida del sistema Sib mayor reglas coral clásica 89
Figura 87. Salida del sistema Sib mayor reglas aleatorias 89
Figura 88. Información Himno de la Alegría introducida al sistema 90
Figura 89. Partitura Himno de la alegría introducida en el sistema 90
Figura 90. Análisis resultante del sistema Himno de la alegría 91
Figura 91. Salida del sistema Himno de la alegría reglas coral clásica 91
Figura 92. Salida del sistema Himno de la alegría reglas aleatorias 92
Figura 93. Información Bacarola introducida al sistema 92
Figura 94. Partitura Bacarola introducida en el sistema 93

Figura 95. Análisis resultante del sistema Bacarola.....93
Figura 96. Salida del sistema Bacarola reglas coral clásica 94
Figura 97. Salida del sistema Bacarola reglas aleatorias 95

ÍNDICE DE TABLAS

Tabla 1. Información contenida en el objeto Note 15
Tabla 2. Datos correspondientes a la clase Nota.....48
Tabla 3. Representación numérica del nombre de las notas 48
Tabla 4. Representación numérica de los acordes y las voces.....49
Tabla 5. Arrays de Strings de la clase Nota 50
Tabla 6. Parámetros de entrada clase Compositor 52
Tabla 7. Parámetros de entrada clase Armonizador 59
Tabla 8. Librerías de jmusic usadas en la clase.....60

1. INTRODUCCIÓN

1.1. Motivación del proyecto

Este proyecto nace con la idea de unir dos ámbitos aparentemente muy alejados como son la música y la ingeniería. Generalmente esta separación viene por parte de los músicos debido a la creencia de que con la tecnología se intenta suplir su trabajo. Nada más lejos del objetivo de este proyecto, que lo que pretende es ser una herramienta de ayuda y experimentación de nuevos estilos mediante Inteligencia Artificial.

En el ámbito estrictamente musical (fuera de los estudios de grabación), la tecnología que se usa como apoyo para la composición es la de editor de partituras, que, aparte de ser una potente herramienta para escribir y guardar la información en formato digital facilitando el intercambio y edición, permite, mediante sintetizadores de sonido, escuchar una aproximación de cómo es la obra. Se habla de aproximación al referirse a la música sintetizada debido a que, aunque el ordenador puede hacer diferencias de intensidad, no puede darle el carácter que un intérprete da a la obra, ya que esa parte es totalmente artística y sentimental.

Por ello se quiere investigar la forma de hacer un módulo de composición musical armónica mediante Inteligencia Artificial en el que se pueda tener una influencia estilística de manera fácil y cuyo resultado pueda ser compartido con otros sistemas, consiguiendo de esta forma la unión entre ambas áreas.

1.2. Objetivos del proyecto

El objetivo de este proyecto es diseñar e implementar un sistema capaz de construir una estructura armónica para una melodía dada según diferentes estilos de composición. Esta estructura será de tipo coral, de forma que la melodía será considerada

la voz más aguda del coro (soprano) y se compondrán otras tres voces correspondientes a contralto (voz femenina), tenor y bajo (voces masculinas). Para lograrlo se usará un sistema experto basado en reglas.

La armonización se llevará a cabo para una agrupación coral que consta de cuatro voces: soprano, contralto, tenor y bajo. Se considerará entrada la voz del soprano y el sistema deberá componer las otras tres voces. Una vez obtenido un resultado se guardará en un archivo MIDI para poder ser reproducido en otros soportes.

Se pretende que el sistema sea fácilmente modificable en la aplicación de diferentes estilos mediante distintos archivos de reglas, de forma que para cambiar de estilo toda la modificación que haya que hacer sea cambiar el archivo de reglas usado para la armonización.

1.3. Estructura de la memoria

Esta memoria consta de 6 capítulos y un apéndice.

El capítulo 1 es la introducción donde se presentan los objetivos del proyecto

El capítulo 2 es el estado del arte. Aquí se hace un repaso de la relación entre la composición musical y las matemáticas, qué son los sistemas expertos, la relación entre la música y la Inteligencia Artificial, y algunos sistemas actuales de composición por ordenador

En el capítulo 3 se describe el armonizador, se repasan algunas definiciones

musicales necesarias y se explica el diseño del sistema.

En el capítulo 4 se implementan dos estilos para ser ejecutados en el armonizador, uno es acorde a las normas de la coral clásica y otro es un sistema aleatorio en el que es el sistema experto el que tiene el control sobre las decisiones.

En el capítulo 5 se presenta la evaluación del sistema mediante tres ejemplos de distintas longitudes y compases. Se obtiene un análisis y una armonización en el caso de las reglas de coral clásica, y una armonización en el caso de las reglas aleatorias.

El capítulo 6 contiene las conclusiones y trabajos futuros basados en el sistema desarrollado.

En el apéndice se incluye el presupuesto del presente proyecto.

2. ESTADO DEL ARTE

En este capítulo se va a hacer una panorámica de la relación que ha existido a lo largo de la historia entre la música y las matemáticas, desde la antigua Grecia hasta los sistemas actuales de composición por ordenador. También se hablará del origen de los sistemas expertos que es en lo que está basado el proyecto, así como de los lenguajes utilizados en este tipo de tecnologías.

El estado del arte de la composición automática basa sus inicios en el siglo XVII cuando surge la necesidad de hacer composiciones de forma rápida, y adaptable a distintas lenguas debido a la expansión de la Iglesia Católica en los nuevos territorios. Como se verá más adelante, este hecho sembró la semilla de la idea de la automatización de la composición musical.

2.1. Composición musical y matemáticas.

Aunque en los orígenes de la música no se puede establecer una unión entre la música y las matemáticas, ésta aparece en el momento en que empieza la teorización de la música.

En la antigua Grecia, Pitágoras (s. VI a.C) y sus seguidores describieron un sistema de ideas que buscaba unificar los fenómenos del mundo físico y del mundo espiritual en términos de números, en concreto, en términos de razones y proporciones de enteros. Creían, por ejemplo, que las órbitas de los cuerpos celestes que giraban alrededor de la tierra producían unos sonidos que armonizaban entre sí produciendo un sonido bello que denominaban la "música de las esferas".

Pitágoras también estudió la naturaleza de los sonidos musicales. La música griega era esencialmente melódica y microtonal, es decir, su escala contenía más sonidos que la escala de 12 sonidos del mundo Occidental.

Él fue el primero que descubrió que existía una relación numérica entre tonos que sonaban "armónicos" y que la música podía ser medida por medio de proporciones de números enteros. Se sabe que el sonido producido al tocar una cuerda depende de variables físicas como son longitud, grosor, tensión, material, etc. Pitágoras descubrió que al dividir la cuerda en ciertas proporciones era capaz de generar sonidos agradables al oído, con lo que según su teoría, el mundo físico y el emocional podían ser descritos con números sencillos y existía una relación armónica entre todos los fenómenos perceptibles [Tiburcio Solís, 2010].

Durante la Edad Media permaneció el concepto de la música como un subconjunto de las matemáticas que se mantuvo hasta el siglo XII, cuando con la Forma Escolástica de Lectio Divina se excluyó de esta disciplina. En este nuevo método se siguen utilizando las matemáticas para calcular los intervalos pero se olvidan los principios pitagóricos y la música se separa de los números.

La relación entre las matemáticas y la música, aunque ya se consideran separadas, se encuentra en dos tipos de situaciones. Por un lado la tradición pitagórica se sigue aplicando, ya que el músico establece en algunas ocasiones un esquema matemático en sus composiciones. Por otro, el músico crea la obra de forma intuitiva usando cánones estéticos, aparentemente sin seguir ningún patrón, y es el matemático el que busca posteriormente una unión entre la composición y las matemáticas.

Un elemento que ha sido usado por compositores como Béla Bartók (1881 - 1945) es la sucesión de Fibonacci. Este autor desarrolló una escala musical basándose en esta sucesión a la que llamó *escala fibonacci*. También en la fuga de *Música para instrumentos*

de cuerda, percusión y celesta aparece dicha serie y la razón áurea.

Analizando autores clásicos encontramos también la relación entre ambas materias. Estudios realizados de la *Quinta Sinfonía* de Beethoven (1770 - 1827) muestran cómo el tema principal de la obra está separado un número de compases perteneciente a la sucesión. También en varias sonatas para piano de Mozart (1756 - 1791) aparece la proporción entre el desarrollo del tema y su introducción, la relación en estas obras es la más cercana posible a la razón áurea.

Estos casos no son situaciones muy extendidas en los músicos de entre el siglo XIII y el XIX, sino que son casos aislados de músicos que como entretenimiento o curiosidad los han usado, o de algunos matemáticos que las han buscado en obras de ese periodo. A partir del siglo XX es cuando se empiezan a buscar nuevas fuentes dentro de las matemáticas [Ríos, 2004].

Un ejemplo de esta búsqueda es Joseph Schillinger (1895 - 1943) que desarrolló un sistema de música que se publicó como el *Sistema de composición musical de Schillinger*. El sistema Schillinger está basado en la geometría y fundamentado en las relaciones de fase de movimientos periódicos simples. Encontró distintas formas de proyectar estas relaciones en el ritmo, pero también en áreas menos obvias como el tono, la escala, los acordes, la progresión armónica e, incluso, en los aspectos semánticos y emocionales de la composición musical [Pérez Ortiz, 2000].

En este proyecto se hace un modelado de sistemas expertos en el ámbito musical basado en la teorización de corales que es habitualmente estudiada en el ámbito académico. Esta teoría se compone de reglas o que toda coral debe cumplir para estar dentro del canon establecido. El compositor más empleado como modelo de referencia en este tipo de composiciones es J.S. Bach (1685 - 1750).

2.2. Sistemas expertos

Alan Mathinson Turing (1910 - 1954) [Encyclopedia Britannica, 2010] fue un matemático y lógico británico que hizo importantes contribuciones a las matemáticas, criptoanálisis, lógica, filosofía y biología, además de nuevas áreas que más tarde se llamaron ciencia computacional, ciencia cognitiva, inteligencia artificial y vida artificial. En 1935 Turing describió una máquina computacional abstracta consistente en una memoria sin límites y un cabezal que se mueve hacia delante y hacia atrás a través de la memoria leyendo y escribiendo símbolo a símbolo. Las acciones de éste cabezal son dictadas por un programa que también está almacenado en la memoria en forma de símbolos. Con este aparato es posible realizar cualquier cálculo que un computador digital sea capaz de realizar. La máquina de Turing se puede considerar como un autómata capaz de reconocer lenguajes formales.

Según la RAE, un sistema experto es un "programa de ordenador o computadora que tiene capacidad para dar respuestas semejantes a las que daría un experto en la materia. Los sistemas expertos pueden mejorar la productividad de un experto en una materia particular mejorando la productividad y ahorrando tiempo y dinero. Son una rama de la Inteligencia Artificial.

Un sistema experto [Samper Márquez, 2010] ["Sistemas Expertos", 2010] está formado por: Base de conocimientos, Memoria de trabajo, Motor de inferencia e Interfaz de usuario.

La base de conocimientos es un tipo especial de base de datos que se usa para la gestión del conocimiento. Se pueden clasificar en dos tipos, las que almacenan conocimiento para ser entendido por un ordenador y las que almacenan conocimiento para ser entendido por personas.

La memoria de trabajo es el lugar de memoria donde se almacenan los datos iniciales del problema que se quiere resolver y donde se guarda todo lo que se va generando (conclusiones y resultados parciales). Es un almacenamiento transitorio de conocimiento para solucionar ese problema en particular.

El motor de inferencia es un módulo de software que usa los datos y el conocimiento para obtener nuevas conclusiones o hechos. Puede seguir dos estrategias, el encadenamiento hacia delante, donde cuando se encadenan las reglas, los hechos pueden utilizarse para dar lugar a nuevos hechos hasta que no puedan obtenerse nuevas conclusiones, o el encadenamiento hacia atrás, donde se propone una solución hipótesis al problema y el algoritmo navega hacia atrás a través de las reglas buscando confirmar dicha hipótesis.

Por último, la interfaz de usuario es la interacción entre el sistema experto y el usuario llevada a cabo mediante lenguaje natural y donde se presentan las conclusiones para ser entendidas por el usuario.

La ventaja de que las partes de un sistema experto aparezcan diferenciadas es que la lógica y los datos aparecen separados, por lo que cualquiera de las partes se puede modificar de forma sencilla cuando haya cambios en un futuro. Además se puede extender a dominios cruzados y multidominios, en el caso de la tecnología usada, permitiendo organizar las reglas en diversos ficheros. El sistema además es escalable y rápido.

Los motores de reglas, que es la tecnología que se va a usar en este sistema, están englobados dentro de los sistemas expertos. La base de conocimientos es una programación declarativa en la que se dice que hacer y no cómo hacerlo. Esto implica que se llega a la solución mediante el uso de reglas.

El algoritmo Rete y sus derivados proporcionan formas muy eficientes de cumplir las reglas en el dominio de los datos. El algoritmo Rete [Rete Algorithm, 2010] fue inventado por Dr. Charles Forgy y documentado en su tesis doctoral en 1978-79. La compilación del algoritmo describe cómo hacer las reglas en la memoria de producción para hacer un eficiente filtrado de datos. La idea es que al principio de la ejecución haya muchos resultados con coincidencias pero según se va iterando haya cada vez menos. Para incrementar la eficiencia del motor sólo se hacen las iteraciones sobre los objetos que hayan presentado coincidencias en la anterior.

Para un sistema de reglas existen dos métodos de ejecución: encadenamiento hacia delante y encadenamiento hacia atrás; los sistemas que implementan ambos se llaman Sistemas de Reglas Híbridos. Un sistema de encadenamiento hacia delante es conducido por los datos y por tanto reaccionario, con hechos que son insertados en la Memoria de Trabajo y como resultado una o más reglas resultan verdaderas programando la ejecución de la Agenda. Se empieza con un hecho que se propaga y se termina con una conclusión. Drools es un motor de encadenamiento hacia delante.

En el encadenamiento hacia atrás se parte de una conclusión que la máquina trata de cumplir. Si no se puede lograr, se busca un objetivo que si que se pueda cumplir, a estas se les llama sub-objetivos que ayudan a resolver algunas partes desconocidas del objetivo que se persigue. Se continúa con el proceso hasta que se logra el objetivo inicial o hasta que no hay más sub-objetivos.

2.2.1. Herramientas y Lenguajes

CLIPS

El origen de *C Lenguage Integrated Production System* (CLIPS) [Riley, 2008] se

remonta a 1984 en *Johnson Space Center de la NASA*. CLIPS es una herramienta de sistema experto que proporciona un entorno completo para la construcción de reglas y de objetos basados en sistemas expertos. Sus características claves son: representación de conocimiento, portabilidad, integración o extensibilidad, desarrollo interactivo, verificación y validación, documentación completa y bajo coste. Actualmente este lenguaje es mantenido independientemente de la NASA como software de dominio público.

El motor de inferencia de CLIPS es de tipo encadenamiento hacia delante, y procesa las reglas mediante el algoritmo Rete. CLIPS soporta dos tipos de paradigma de programación, la programación imperativa como son C y Ada, y la programación orientada a objetos. [What is clips?, 2010]

```
(defrule example-rule  "This is an example of a simple rule"
   (refrigerator light on)
   (refrigerator door open)
   =>
   (assert (refrigerator food spoiled)))
```

Figura 1. Regla en CLIPS

La figura 1 muestra un ejemplo de una Regla en CLIPS en el que se comprueba si la luz de un frigorífico está encendida, si se cumple comprueba si la puerta está abierta, en caso de que esta comprobación sea también afirmativa se inserta un mensaje en la memoria de trabajo que avisa de que la comida del frigorífico está estropeada.

Jess

Jess [Friedman-Hill, 2008] es un motor de reglas y entorno de programación escrito íntegramente en Java de Sun Microsystems. Usando Jess se puede construir un software Java con capacidad de "razonamiento" usando conocimiento proporcionado en forma de reglas declarativas. Jess utiliza una versión mejorada del algoritmo Rete para procesar las

reglas. El motor de inferencia que usa es el de tipo encadenamiento hacia delante.

En Jess las reglas pueden ser implementadas mediante el lenguaje de reglas de Jess y mediante XML. Se puede programar usando la API de Java mediante la biblioteca javax.rules. Este lenguaje se puede implementar en cualquier editor de textos, aunque también incluye un plug-in para Eclipse que facilita esta labor.

```
Jess>(defrule welcome-toddlers
    "Give a special greeting to young children"
    (person {age < 3})
    =>
    (printout t "Hello, little one!" crlf))
```

Figura 2. Regla Jess

En la figura 2 se presenta una regla en lenguaje Jess a la que se le da el nombre “welcome-toddlers” (bienvenida a los niños) en la que se buscan los objetos de la clase persona cuyo valor del atributo edad es menor a 3 y se imprime un mensaje de saludo por pantalla.

Drools

Drools [Drools Expert User Guide, 2010] es un sistema de reglas (Business Rule Management System) con un motor de inferencia de tipo encadenamiento hacia delante basado en un motor de reglas, más conocido como un sistema de reglas de producción mediante la aplicación del algoritmo Rete.

Drools implementa el estándar JSR-94 para el motor de reglas y el entorno de desarrollo, mantenimiento y ejecución de las políticas de negocio en una organización, una aplicación o un servicio.

El proyecto Drools fue iniciado en 2001 por Bob McWhirte y registrado en SourceForge. Drools 1.0 nunca se publicó debido a la limitación producida por el enfoque de búsqueda lineal. Se publicó directamente Drools 2.0 basado en el algoritmo Rete.

```
rule
    when
        Cheese( $cheddar : name == "cheddar" )
        $person : Person( favouriteCheese == $cheddar )
    then
        System.out.println( $person.getName() + " likes cheddar" );
end
```

Figura 3. Regla Drools

En la Figura 3 se muestra una regla en la que se tienen dos objetos, uno llamado Queso y otro llamado Persona. En esta regla se buscan las personas cuyo valor de queso favorito es Cheddar y se imprime por pantalla el contenido del atributo nombre del objeto (en este caso de la clase Persona). Las reglas en Drools son una estructura de dos partes en la que se usa lógica de primer orden para razonar sobre la presentación del conocimiento.

Drools implementa y extiende el algoritmo Rete. La implementación de Drools de este algoritmo se llama ReteOO, que significa que la implementación es para sistemas orientados a objetos.

Las reglas se almacenan en la Memoria de Producción y los hechos (facts), que en el Motor de Inferencia se derivan de las reglas, son guardados en la Memoria de Trabajo. Los hechos son añadidos a la Memoria de Trabajo donde deben ser modificados o eliminados. En un sistema con un gran número de reglas y hechos, puede ocurrir que varias reglas cumplan las condiciones de ejecución al mismo tiempo, se dice que estas reglas están en conflicto. La Agenda gestiona el orden de ejecución de las reglas en conflicto usando una estrategia de Resolución de Conflictos.

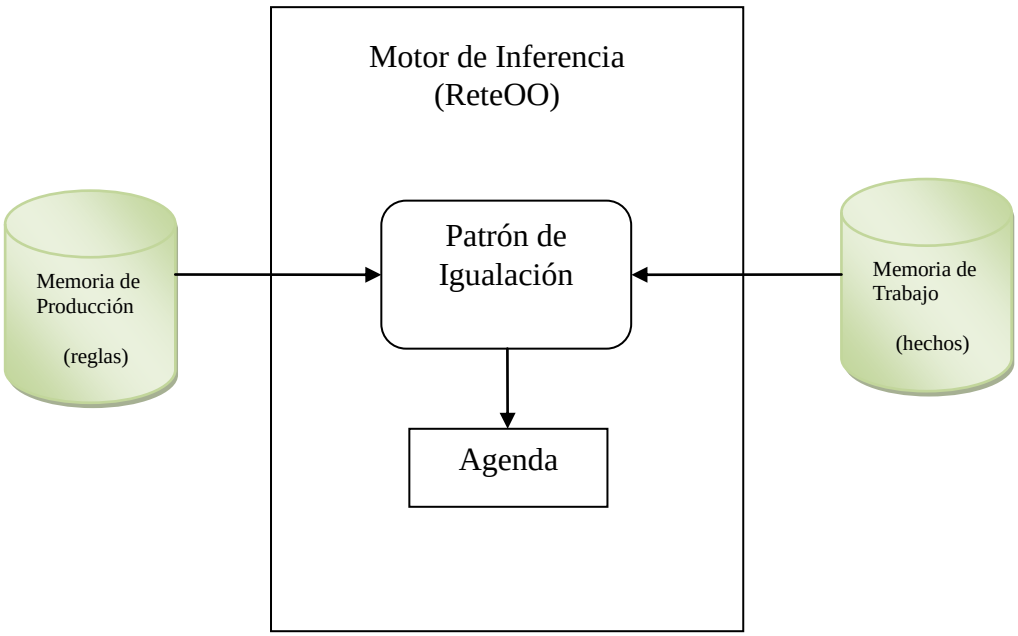

Figura 4. Vista de alto nivel de un Motor de Reglas

Drools tiene un sistema nativo de lenguaje de reglas. El formato es muy permisivo en términos de puntuación, y soporta lenguajes naturales y de dominio específico a través de expansiones que permite al lenguaje adaptarse al dominio del problema. Un archivo de reglas es típicamente un archivo con extensión .drl. En un archivo DLR se pueden tener múltiples reglas, queries y funciones así como también algunas declaraciones de fuentes como importaciones, globales y atributos que son asignados y usados por las reglas y las queries. También se pueden distribuir las reglas por distintos archivos lo que puede facilitar el manejo de números muy elevados de reglas. Un archivo DLR es un archivo de texto.

```
package package-name

imports

globals

functions

queries

rules
```

Figura 5. Composición de archivo de reglas

El orden de la declaración de elementos no es importante a excepción del nombre del paquete que tiene que ser declarado como primer elemento de un archivo de reglas. El resto de elementos son opcionales y solo se utilizan en caso de ser necesario.

```
rule "name"
    attributes
    when
        LHS
    then
        RHS
end
```

Figura 6. Estructura de una regla

En la Figura 6 se representa la estructura de una regla. Las comillas dobles para el nombre son opcionales al igual que los atributos que son indicaciones de cómo una regla debería funcionar. LHS (Left Hand Side) es la parte condicional de la regla y RHS (Right Hand Side) es el código que debe ser ejecutado.

El LHS está formado por cero o más elementos condicionales. Si está vacío se interpreta como `eval(true),` que significa que la condición siempre se cumple. Se activará una vez, cuando una nueva sesión de la memoria de trabajo sea creada. El RHS es conocido también por el nombre de consecuencia o acción de una regla. Esta parte debe contener una lista de acciones que hay que ejecutar. El principal propósito del RHS es insertar, eliminar o modificar la memoria de trabajo.

JMusic

En el proyecto se usará una librería de Java llamada JMusic. JMusic [Brown&Sorensen, 2010] es un proyecto diseñado para proporcionar a compositores y desarrolladores de software paquete Java de herramientas para el procesado de audio y composición.

Las aplicaciones Java pueden ser escritas usando componentes Jmusic. Estos componentes incluyen una estructura de datos musicales con métodos de modificación y traducción de clases asociados. También tienen algunos elementos de interfaz gráfica de usuarios. La estructura de datos es:

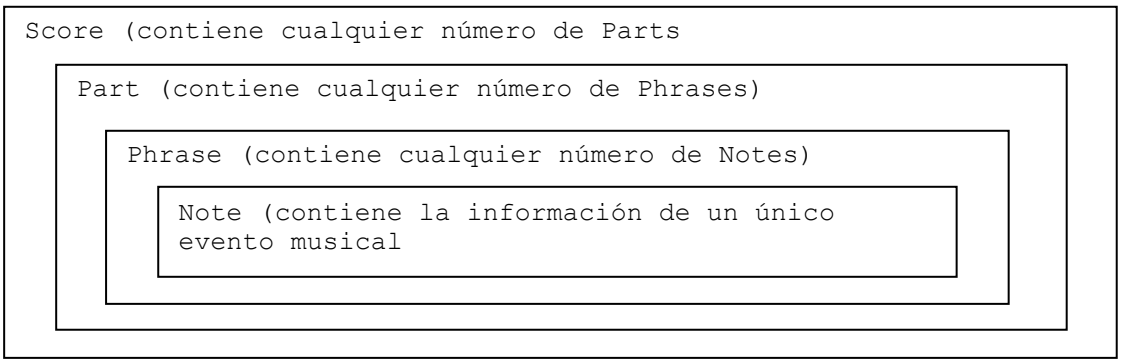

Figura 7. Estructura de datos de JMusic

La clase `Score` representa el nivel superior de la estructura de datos y contiene un vector de `Parts` y un título. La clase `Part` contiene un vector con `Phrases`. Puede tener además un título (por ejemplo “Violín I”), un canal y un instrumento. La clase `Phrase` puede ser interpretada como voz. Una única parte de piano puede tener múltiples voces como la mano izquierda y la mano derecha. Los objetos `Phrase` contienen una lista de `Notes`. Estos objetos se pueden añadir, eliminar y mover. El objeto `Note` es la estructura de las notas usada en JMusic. La información que contiene es:

Pitch	El tono de la nota
Dynamic	El volumen sonoro de la nota
RythmValue	El valor rítmico de la nota
Pan	La posición de la nota en el espectro estéreo
Duration	La longitud en milisegundos de la nota
Offset	Desviación del comienzo ‘normal’ de la nota (comienzo en anacrusa)

Tabla 1. Información contenida en el objeto Note

MIDI (Musical Instrument Digital Interface)

Hay muchos tipos de dispositivos que usan MIDI [Tutorial: The technology of MIDI, 2010], desde teléfonos móviles a teclados musicales u ordenadores. Este lenguaje describe el proceso de reproducción musical de forma similar a una partitura a través de mensajes. Hay mensajes MIDI que describen qué notas deben ser tocadas, durante cuánto tiempo, así como el tempo (velocidad) de la obra, qué instrumentos tocan, sus volúmenes relativos, etc.

Dentro del estándar MIDI se encuenta el estándar MIDI Files, que tiene dos tipos de archivos: el archivo de Tipo 1, donde las partes son guardadas en distintas pistas dentro de la misma secuencia, y el de Tipo 0, donde todo es guardado en una única pista. Un archivo MIDI no solo puede contener información de reproducción (canales, longitudes, tonos…), sino que puede también contener información adicional de configuración (tempo, instrumentos de cada pista…) e información de la canción (copyright, compositor, fecha de composición…). Este tipo de archivo además proporciona la posibilidad de escribir la letra correspondiente a la melodía y también información de expresión como acentos y ligaduras, por lo que le hace un estándar muy extendido en la edición de partituras.

2.3. Música e Inteligencia Artificial

Hasta el siglo XVII no aparece, ideado por Athanasius Kircher (1601 - 1680), el primer sistema algorítmico de composición musical, el Arca Musarithmica [Bush & Kassel, 2006] (sistema creado para componer de forma mecánica). Famoso es también el Musicalishcher Würfelspier, conocido como el juego de dados de Mozart. En 1821, Winkel, inventó el Componium, un aparato mecánico que genera variaciones sobre un tema pre-programado. Hacia el año 1920, Arnold Shoenber expuso unas técnicas

dodecafónicas que han tenido una vital transcendencia en la música culta del siglo XX como la primera aplicación masiva de un sistema de composición algorítmico. Otros compositores, como Erik Satie, han afirmado por su parte trabajar de forma totalmente sistematizada. Aunque las pequeñas piezas de éste compositor francés figuran entre las más "emotivas" del siglo XX.

En 1941, Joseph Schillinger, prediciendo el uso de los ordenadores en la composición musical, intentó desarrollar una teoría musical que pudiese ser útil para su uso con cerebros electrónicos. En referencia al término "Inteligencia artificial" en su forma más general, y considerando como tal, todos los trabajos llevados a cabo en el campo de la composición asistida por ordenador, el pionero fue el químico y compositor Lejaren Hiller (1924 - 1994), que en 1955 inició sus trabajos de música algorítmica en la universidad de Illinois. En 1957, Hiller publicó la *Suite Illiac*, que es la primera composición musical íntegramente realizada por un ordenador, para la que combinó la utilización de cadenas de Markov y otros conceptos tomados de la teoría de la información, con normas musicales adaptadas del "Cantus Firmus".

Winograd (1946) realiza un estudio menos ambicioso, basado en la gramática sistémica de Halliday. Su programa lleva a cabo el análisis armónico de una pieza, realizando el etiquetado de acordes, similar al que se realiza en cualquier curso de armonía. Constituye uno de los primeros sistemas de análisis eficaz.

A principios del siglo XX, Heinrich Schenker (1868 - 1935), estableció un sistema transformacional de análisis musical. Enfocó el problema desde un punto de vista estructural, dividiendo toda pieza musical en varios niveles. El primero, el orden cercano o superficial, está compuesto por los intervalos existentes entre las notas, el segundo nivel está compuesto por los mínimos motivos musicales que están constituidos por 2, 3 ó 4 notas. Estos niveles no se resuelven siempre de forma unívoca. Así se llega a la estructura profunda o Ursatz, que describe la composición, y que engloba todas sus diferentes partes y sus repeticiones. Según su autor, todas las buenas composiciones

comparten en su estructura profunda, unos pocos ursatze posibles. Schenker utiliza ya para su análisis, una representación en forma de árbol.

Iannis Xenakis (1922 - 2001) fue uno de los primeros compositores que comprendieron el potencial de los ordenadores en la resolución de nuevos problemas musicales. Para él la música es "la armonía del mundo, homomorfizada por el pensamiento actual", y también la forma de expresar la inteligencia mediante sonidos desprovistos de significado semántico, lo que le sitúa en una posición pitagórica [Jordá, 2010].

2.3.1. Arca Musarithmica

El Arca Musarithmica [Bush & Kassel, 2006] no se trata exactamente de un sistema automático, sino que es un sistema creado para poder componer de manera mecánica. Para lograrlo Kircher, utilizando técnicas combinatorias, diseñó un conjunto de tablillas (*tablae melotacticae*) que contienen acordes a cuatro voces siguiendo las convenciones del bajo continuo en secuencias preparadas para encajar cada una en un metro poético específico, con varias estructuras rítmicas para aplicar a las notas. Está preparado tanto de división binaria como ternaria.

La mecánica de la composición consiste en encadenar diversas secuencias de acordes cifrados (*musarithmus*) respetando las características de medida y ritmo (distribución de acentos) del texto escogido.

Kircher diseñó este sistema de los fundamentos rítmicos, armónicos y retóricos de la música práctica para ayudar en la labor misionera de los jesuitas (congregación a la que pertenecía), que debían utilizar la música para adoctrinar a los nuevos pueblos, pero

se encontraban con el problema de adaptar los textos indígenas a las melodías litúrgicas de la Iglesia de Roma. Cualquier texto en cualquier lengua y con cualquier contenido podría ser, de este modo, trasladado a la música sin mucha dificultad.

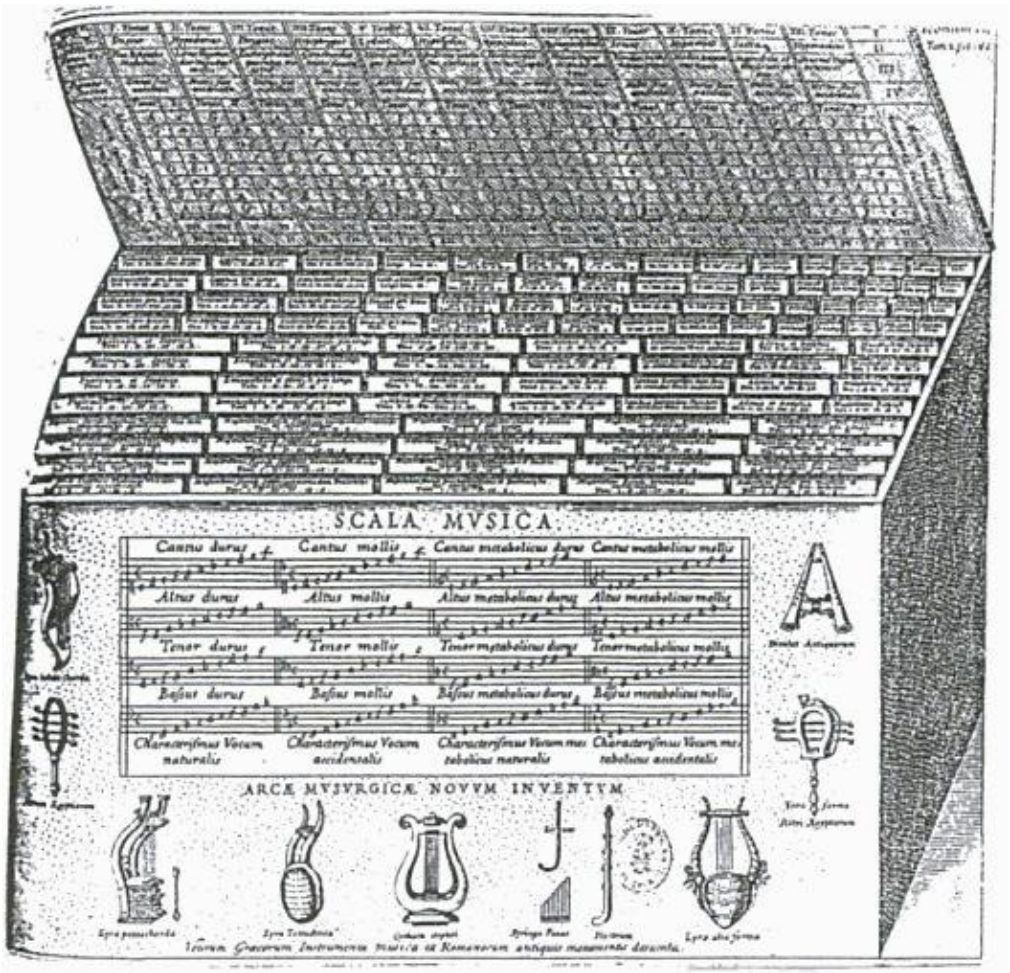

Figura 8. Diseño original del arca musarithmica

Figura 9. Arca Musarithmica

2.3.2. Musikalishcher Würfelspier

El Musicalishcher Würfelspier [Juego de los dados de Mozart, 2010] es una singular creación artística que no es una pieza para piano, sino un generador de valses. No es una partitura para un vals de 16 compases, sino que es un sistema, que apoyado en el azar, puede generar un número mucho mayor de valses diferentes de 16 compases cada uno.

Mozart escribió 176 compases numerados del 1 al 176 y los agrupó en 16 conjuntos de 11 compases cada uno. El procedimiento para generar un vals consiste en que cada compás del 1 al 16 se selecciona con unos dados, del correspondiente conjunto de 11 compases.

	I	II	III	IV	V	VI	VII	VIII
2	96	22	141	41	105	122	11	**30**
3	32	6	128	63	146	46	134	81
4	69	95	158	13	153	55	110	24
5	40	17	113	85	161	2	159	100
6	148	74	163	45	80	97	36	107
7	104	157	27	167	154	68	118	91
8	152	60	171	53	99	133	21	127
9	119	84	114	50	140	86	169	94
10	98	142	42	156	75	129	62	123
11	3	87	165	61	135	47	147	33
12	54	130	10	103	28	37	106	5

	IX	X	XI	XII	XIII	XIV	XV	XVI
2	70	121	26	9	112	49	109	14
3	117	39	126	56	174	18	116	83
4	66	139	15	132	73	58	145	79
5	90	176	7	34	67	160	52	170
6	25	143	64	125	76	136	1	93
7	138	71	150	29	101	162	23	151
8	16	155	57	175	43	168	89	172
9	120	88	48	166	51	115	72	111
10	65	77	19	82	137	38	149	8
11	102	4	31	164	144	59	173	78
12	35	20	108	92	12	124	44	131

Figura 10. Tablas del juego de dados de Mozart

En el encabezado, en números romanos aparece el número del compás e identificando cada una de las filas aparece un número entre 2 y 12 que corresponde a la suma de las caras de dos dados que deben ser lanzados para definir en cada compás, cuál es el elemento que deberá incluirse en la partitura. La obra aparece publicada por primera vez en la Edición de J.J. Hummel, Berlín-Amsterdam, 1793.

2.3.3. Componium

Es un órgano mecánico que compone por sí mismo, inventado en 1821 por Dietrich Nikolaus Winkel (1777 - 1826) [Bush & Kassel, 2006]. El tamaño del componium es de 2.7 x 1.42 x 0.57 metros. Es una máquina que utiliza controladores aleatorios para producir variaciones virtuales infinitas de un tema. Está formado por dos órganos mecánicos por un lado, y una “máquina de composición” (componium) por otro. Tiene dos cilindros que usan alfileres y ganchos para la frecuencia y la duración, respectivamente, del sonido. El sonido se produce cuando el alfiler y el gancho alcanzan un nivel que abre un tubo muy parecido a cualquier órgano de cilindros mecánico.

La máquina de composición mueve los cilindros en diferentes direcciones independientemente unas de otras. Por ejemplo, un tema de ocho compases necesita cuarenta filas de combinaciones de alfiler y gancho (cada fila tiene dos medidas). Los cilindros contienen también siete variaciones. El total de las ocho variaciones (original y las siete mencionadas anteriormente) es la base para cambiar entre las filas como dicte la máquina de composición creando cualquier número de sucesiones de filas.

El componium se mostró en Londres y París, pero el interés del público no respondió a las necesidades económicas de Winkel, que murió en la pobreza. El componium se desechó y prácticamente se desintegró. Fue adquirido por Victor-Carles Mahillon, el primer conservador del Museo de Instrumentos Musicales de Bruselas. El componium fue restaurado a mediados de la década de los 60, aunque en la actualidad vuelve a encontrarse sin funcionamiento.

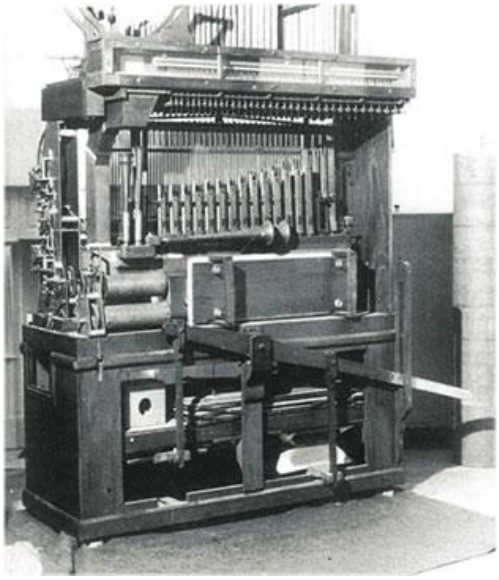

Figura 11. Fotografía de un componium

2.3.4. Illiac Suit

El cuarteto de cuerda Illiac Suit [Sandred & Laurson & Kuuskankare, 2010] es el primer ejemplo de procesos aleatorios en la composición de música por ordenador. La idea general es usar reglas de selección para aceptar o rechazar las notas y ritmos generados de forma aleatoria. Se usaron distribuciones de probabilidad y procesos de Markov en esta obra.

Se usan tablas de probabilidad para controlar la distribución de los intervalos melódicos de las cuatro voces. El ritmo en el cuarto movimiento se fijó a un ostinato (secuencia rítmica que se repite durante toda la obra o movimiento) de corcheas, el ordenador solo fue usado para decidir que notas se daban a esas corcheas. Las tablas de probabilidad se cambiaron cada dos compases para este movimiento.

La Illiac Suit tiene ese nombre en honor al ordenador con que fue creada, ILLIAC I (Illinois Automatic Computer). ILLIAC I fue creada en 1952 en la universidad de Illinois y fue la primera con fines educativos. Cuando fue creada tenía mayor capacidad computacional que todos los ordenadores de los Laboratorios Bell.

Figura 12. Fotografía del ordenador ILLIAC I

La entrada del ILLIAC I [Bohn, 2010] era de 100 caracteres por segundo y la de salida de 60 caracteres por segundo. La programación de la máquina se hacía a través de órdenes que ocupaban el espacio de una palabra de 40 bits.

El ILLIAC I fue el primer ordenador disponible en la Universidad de Illinois. Trabajar en ese ordenador se consideraba de vanguardia debido a la escasez de ordenadores y la calidad de la máquina.

2.4. Sistemas actuales de composición por ordenador

En este apartado se describirán sistemas de composición realizada por ordenador que se encuentran actualmente en el panorama de la música y la tecnología.

2.4.1. Proyecto EMI de David Cope

EMI (Experiments in Musical Intelligence) [Gómez-Zamalloa Gil, 2010] es un sistema que analiza partituras existentes buscando patrones musicales recurrentes llamados signaturas, que posteriormente son recombinados formando nuevas composiciones acordes al estilo del compositor analizado.

Este sistema se basa en cuatro fases: análisis (intenta determinar el carácter y función de determinados grupos de notas), identificación de patrones (para construir el esqueleto básico de la nueva composición), deconstrucción (las partituras de entrada se dividen en pequeños segmentos musicales) y reconstrucción (se combinan los elementos para crear una pieza musical nueva).

Para componer los motivos entre signaturas, EMI usa un analizador implementado como una red aumentada de transiciones (ATN) que creará el núcleo de la base de reconstrucción y busca restricciones o propiedades existentes en obras del mismo compositor.

2.4.2. Designing Music

Designing Music [Robles, 2010c] es una técnica de creación musical desarrollada por Luis Robles, compositor, profesor de armonía, análisis y composición e Ingeniero por la Universidad Politécnica de Madrid.

La idea original de este proyecto fue cómo poder plasmar en la música sentimientos de la misma forma que los pintores lo hacen en sus obras mediante colores. Designing Music es un programa de ordenador que inicialmente el desarrollador concibió

para uso personal pero que finalmente publicó bajo la licencia de Creative Commons.

Designing Music está formado por dos módulos, el compositor, que genera la forma y la temática del contenido de la obra, y el armonizador, que crea una armonización para el resultado devuelto por el compositor.

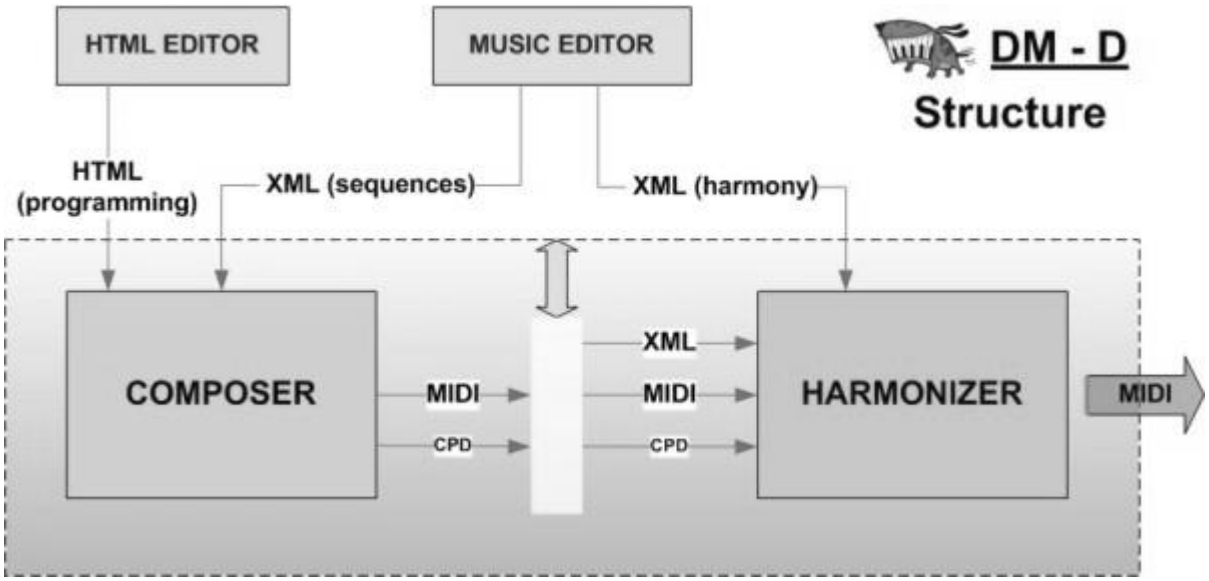

Figura 13. Estructura de DM-D

Este sistema requiere de dos programas externos para funcionar, un editor de HTML para escribir la hoja de programación para el compositor, y un editor de música, ya que el usuario puede coger secuencias externas de música para crear ideas musicales. Además el armonizador requiere que el usuario escriba los acordes para dar color (armonía) a la obra.

La comunicación entre módulos se hace mediante archivos que pueden ser editados o revisados en cualquier momento. Las operaciones a seguir para la ejecución del programa son: escribir la hoja de programación para el compositor, ejecutar el compositor, escribir la hoja de armonía y ejecutar el armonizador.

2.4.3. SICOM

SICOM [Gómez-Zamalloa Gil, 2010] es un sistema basado en el análisis musical, CBR (Case Based Reasoning) y técnicas de planificación. Las nuevas soluciones se obtienen transformando y extrapolando el conocimiento derivado de los análisis musicales previamente realizados por expertos. Cada análisis se representa como un caso y se divide la pieza musical según la siguiente jerarquía:

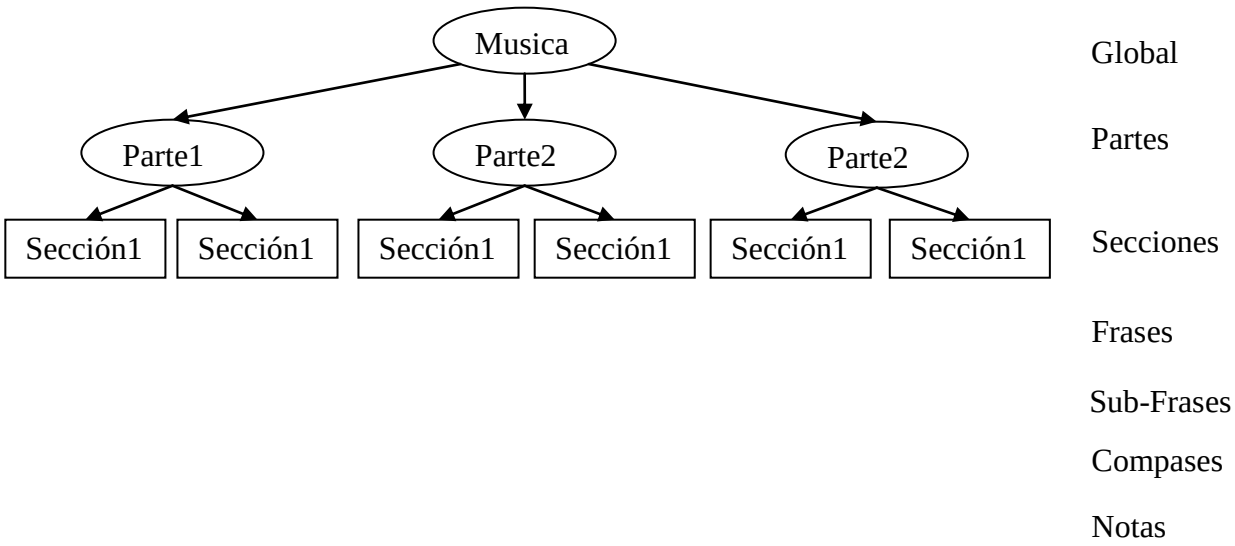

Figura 14. Jerarquía del análisis llevado a cabo por SICOM

Además de las relaciones jerárquicas puede haber relaciones horizontales entre componentes. Por ejemplo, en la Figura 15 se indica que la sección A de la parte 1 es la misma que la sección A de la parte 2 pero transportada.

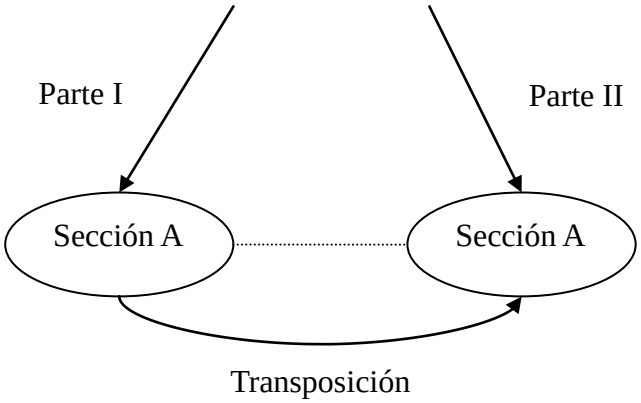

Figura 15. Relación horizontal entre estructuras

En SICOM cada una de estas estructuras en forma de árbol junto con las relaciones horizontales entre componentes forma un caso, donde se extrae a continuación información en el ciclo CBR.

2.4.4. Finale

Finale es un programa comercial que tiene muchas funcionalidades, entre las que se incluyen edición de partituras y un asistente para crear acompañamientos para banda, orquesta y una sola voz.

A través del asistente se va guiando al usuario para elegir el tipo de agrupación y los instrumentos. Al ser también editor de partituras, se puede usar para sacar las partes de cada instrumento por separado. También reproduce los archivos MIDI con lo que se puede usar para escuchar la composición o para que el músico pueda ensayar su parte escuchando el resto de voces. También dispone de otra herramienta llamada Studio View cuya finalidad es enseñar composición al usuario. [Finale, 2010]

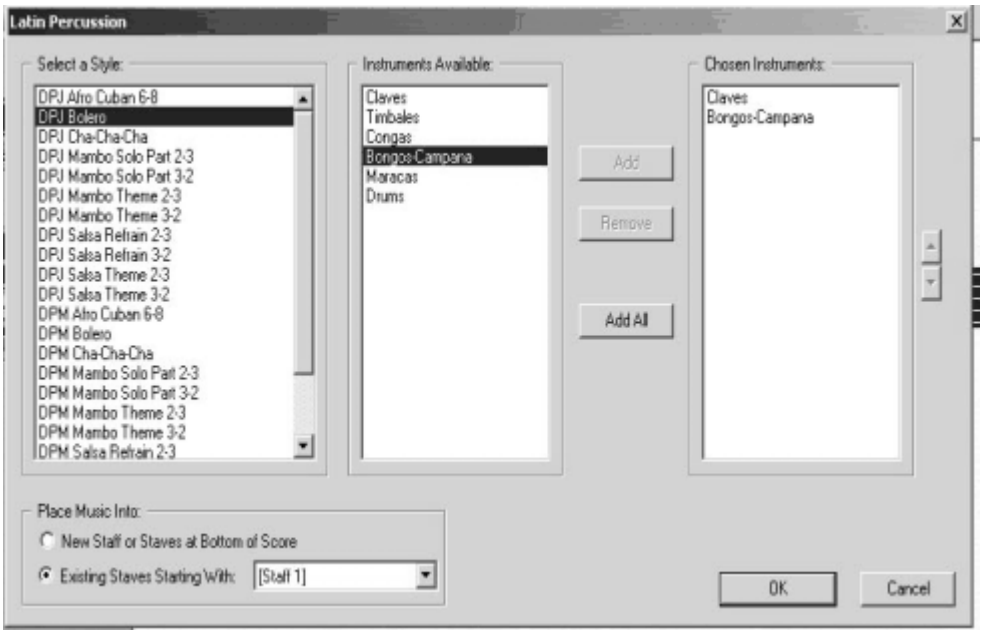

Figura 16. Captura de Finale de composición de percusión

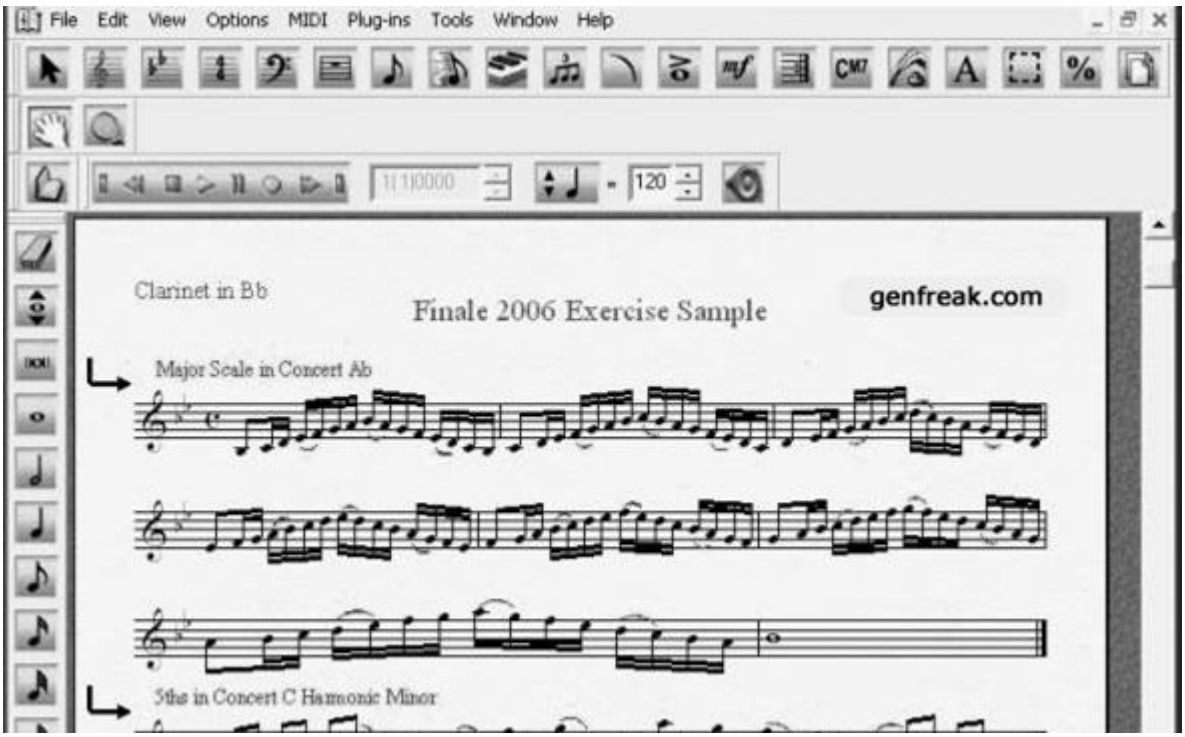

Figura 17. Captura de pantalla de Finale

3. ARMONIZADOR

En este capítulo se hará una breve introducción a los conceptos musicales necesarios para el sistema, que también se describirá, y las reglas musicales que posteriormente se han desarrollado en la implementación específica. El proyecto consiste en un sistema que recibe una melodía MIDI y aplicando un archivo de reglas también proporcionado, devuelve la melodía armonizada.

3.1. Definiciones

A continuación se van a definir y explicar algunos conceptos musicales que son necesarios para la composición musical clásica. Las definiciones se extraen del diccionario de la Real Academia de la Lengua Española, del documento “Estructuras del lenguaje musical”, del documento “6. Intervalos, escalas y tonalidad”, del documento “Armonía”, y del documento “Teoría Musical”.

Acorde: Conjunto de tres o más sonidos diferentes combinados armónicamente. Un sonido equivale a una nota en el plano de la escritura musical.

Acorde de tríada: Conjunto de tres sonidos diferentes combinados armónicamente.

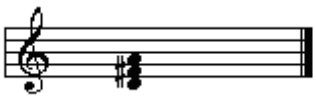

Figura 18. Acorde de tríada de Re Mayor. Compuesto por Re, Fa sostenido y La

Acorde cuatríada: Conjunto de cuatro sonidos diferentes combinados armónicamente. Se conocen como acordes de séptima.

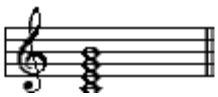

Figura 19. Acorde de cuatríada de Do Séptima. Compuesto por Do, Mi, Sol y Si

Acorde mayor: formado por tres notas, la fundamental o tónica, una tercera mayor y una quinta. La tercera mayor se encuentra a una distancia de dos tonos y la quinta de tres tonos y medio.

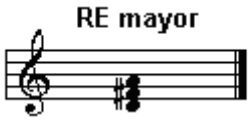

Figura 20. Acorde de Re Mayor. Compuesto por Re, Fa sostenido y La

Acorde menor: formado por tres notas, fundamental o tónica, una tercera menor y una quinta. La tercera menor se encuentra a una distancia de un tono y medio y la quinta de tres tonos y medio.

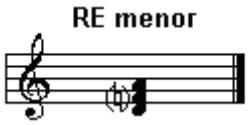

Figura 21. Acorde de Re menor. Compuesto por Re, Fa y La

Secuencia de acordes: consecución de acordes.

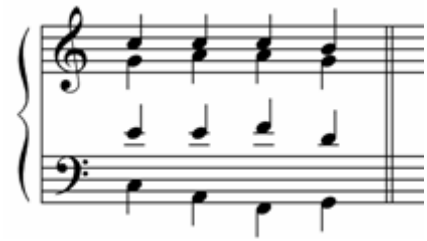

Figura 22. Secuencia de acordes

Enlace: disposición de las notas de dos acordes consecutivos.

Intervalo: es la distancia que existe entre dos notas de distinta altura. Esta distancia se cuenta por los tonos y semitonos existentes entre esas dos notas.

Coral: Composición vocal armonizada a cuatro voces (soprano, contralto, tenor y bajo), de ritmo lento y solemne, ajustada a un texto de carácter religioso y que se ejecuta

principalmente en iglesias protestantes. El compositor de referencia de este género es J.S. Bach (1685 - 1750).

Carácter musical: es la propiedad por medio de la cual se manifiestan los sentimientos en la música.

Escala: sucesión de sonidos consecutivos de un sistema (tonalidad) que se suceden regularmente en sentido ascendente o descendente, y todos ellos con relación a un solo tono que da nombre a toda la escala. Las escalas mayores siempre siguen la secuencia tono-tono-semitono-tono-tono-tono-semitono independientemente de la nota en la que se empieza. Las escalas menores siguen la secuencia tono-semitono-tono-tono-semitono-tono-tono.

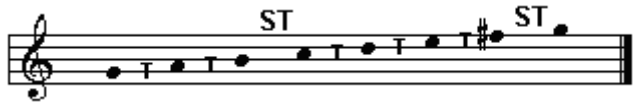

Figura 23. Escala en Sol mayor

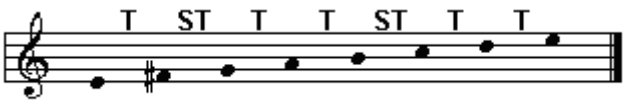

Figura 24. Escala de Mi menor

Armonía: es la disciplina de la música en la que se estudia las combinaciones simultáneas de sonidos, llamadas acordes, para reproducir sensaciones sonoras.

Tipos de movimiento entre dos voces:

- Contrario: las voces se mueven en direcciones opuestas
- Directo: Las voces se mueven en la misma dirección
- Oblicuo: Una voz se mueve mientras otra permanece quieta

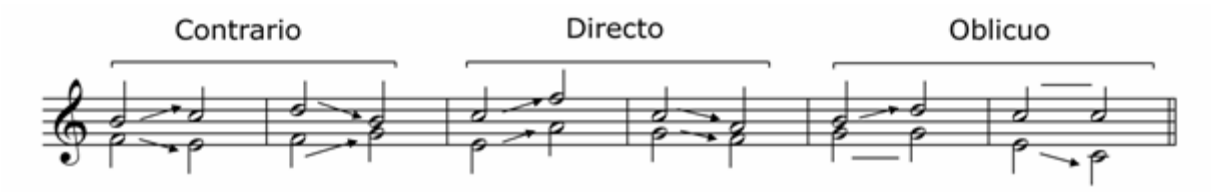

Figura 25. Tipos de movimientos entre dos voces

Tesitura: Altura propia de cada voz o cada instrumento.

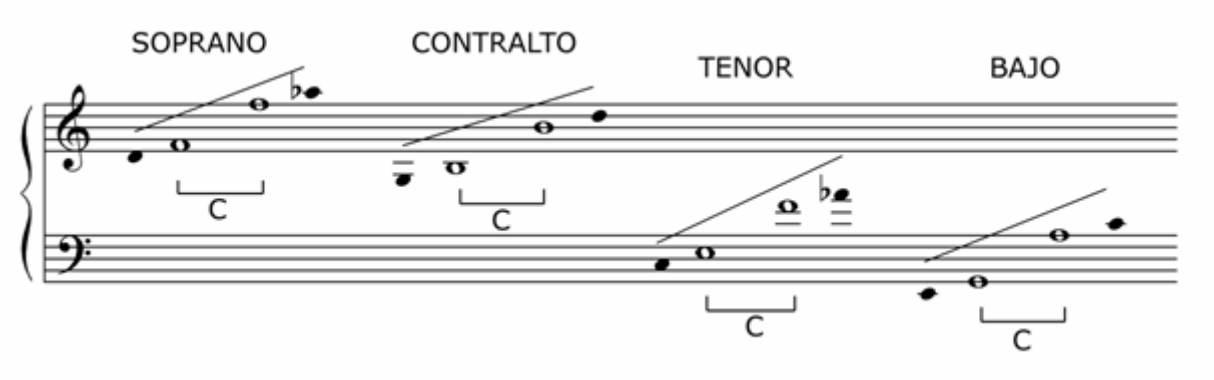

Figura 26. Tesituras de las voces de los coros. Las blancas representan la tesitura que se emplea en la composición de corales y las negras la tesitura a la que algunos cantantes de esa voz pueden llegar

Armonizar una melodía: escribir una armonía para una melodía. En corales implica escribir las tres voces inferiores a la melodía indicada por la voz soprano: contralto, tenor y bajo.

Claves: En música existen 7 claves.

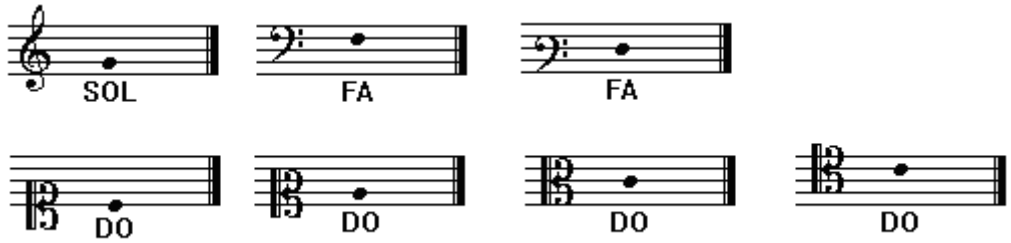

Figura 27. Representación de todas las claves musicales: Sol, Fa en cuarta, Fa en tercera, Do en primera, Do en segunda, Do en tercera y Do en cuarta

En la nomenclatura de coral se usan la clave de sol para las voces contralto y soprano y la de Fa en cuarta para las voces de tenor y bajo. Las distintas claves sirven para representar distintas tesituras dentro del pentagrama sin tener que recurrir sistemáticamente a las líneas adicionales.

Compás: Es la división de la grafía musical en una serie de porciones iguales. Dividen al pentagrama en partes iguales y agrupan a un fragmento de notas musicales. Según la cantidad de partes que contienen se puede clasificar en binarios, ternarios y

cuaternarios. El compás está dividido a su vez en partes iguales llamados tiempos. Están separados por líneas divisorias.

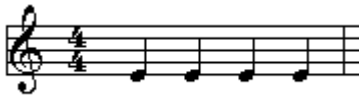

Figura 28. Compás de 4/4

El denominador del compás es la medida que se toma de referencia para su longitud y el numerador es el número de valores de referencia que caben en el compás.

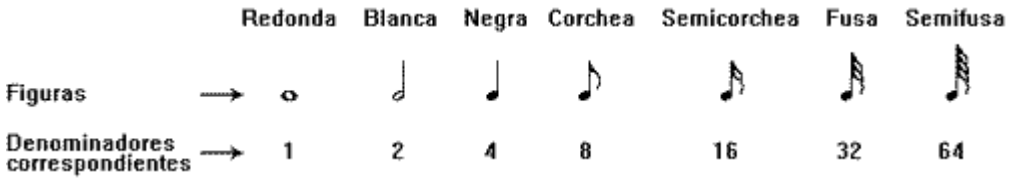

Figura 29. Correspondencia entre figuras y denominadores

Valor de las notas: es la determinación de la duración de los sonidos de forma relativa entre sí. Los silencios son las pausas sonoras entre los sonidos. El valor en tiempo depende también de la velocidad del pulso

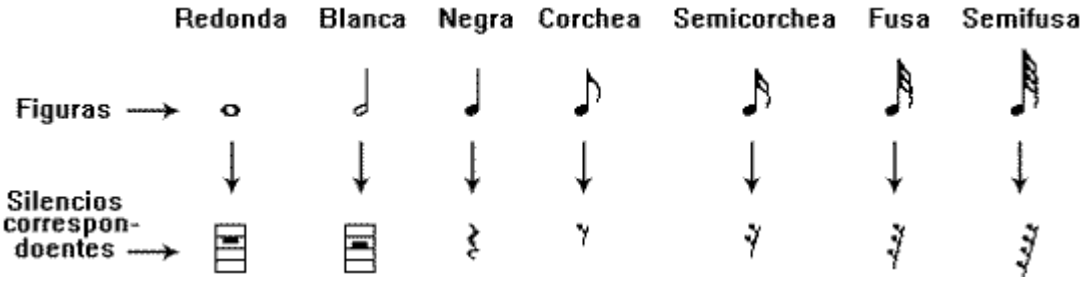

Figura 30. Correspondencia entre figuras y silencios

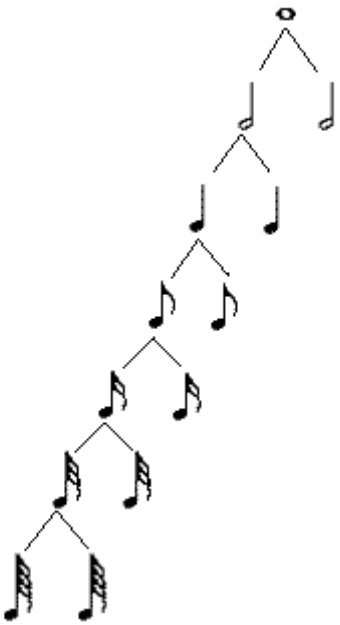

Figura 31. Correspondencia de valores relativos

Tonalidad: nota central llamada tónica en la que se basa la ordenación de las notas pertenecientes a la misma.

Grado: número de representación en la escala de la nota sobre la que se construye el acorde de armonización.

Reglas de armonía clásica

El armonizador está basado en las reglas de la armonía clásica que son las estudiadas en el ámbito educativo. Existen múltiples tratados de armonía desarrollados tanto por profesores (Joaquín Zamacois, Enric Herrera…) como por famosos compositores (Rimsky Korsakov entre ellos). Estás reglas son la forma que se consiguió de listar lo que según los cánones de belleza suena bien. No son tanto unas reglas fijas como una descripción o una indicación de lo que se puede hacer y de lo que no. Según esta forma de composición se deben cumplir las normas, aunque siempre se puede hacer alguna excepción.

En el sistema se tendrán dos tipos de notas, las que aún son provisionales (posibles) ya que en un principió se introducirán todas las posibilidades correspondientes a la nota a armonizar, y las que ya son definitivas (finales), que son las que tras la aplicación de reglas se ha llegado a la solución de que es la más idónea para la posición.

En la aplicación que se desarrollarán las siguientes reglas musicales de la armonía clásica [Robles, 2010a] [Robles, 2010b] [Zamacois, 1945]:

- Tesituras: Cada voz del coro se encuentra dentro del rango correspondiente a un cantante de esa tesitura. Eventualmente se puede ampliar este rango, pero un cantante medio es posible que no pueda cantarlo, por tanto, en este desarrollo se ha optado por limitar los rangos a los extremos de un cantante medio de cada voz. Estos límites son: Contralto (B3 - B4), Tenor (E3 - F4) y Bajo (G2 - A3). Las de la Soprano (F4 - F5) no se tienen en cuenta por ser la entrada del programa y no poder ser modificada. Las tesituras del coro son mostradas en la Figura 26.

  Para trasladar esto al sistema se hará mediante el valor de las notas en MIDI. Las notas en MIDI se representan mediante números enteros, por tanto se acotarán las voces eliminando los valores que se encuentren fuera del rango permitido. Esta eliminación se hará de forma independiente en las tres voces debido a que cada una tiene un rango distinto.

- Duplicación de notas: Al hacerse la armonización con acordes de tríadas al menos una de las notas del acorde debe ser duplicada.

  No se pondrán restricciones a cuál debe ser duplicada ya que no hay una regla que obligue a que sea una en concreto. Algunos compositores no incluyen todas las notas.

  Como se ha indicado con anterioridad, los acordes de tríada constan de tres notas, el caso de Do mayor, el primer acorde está formado por Do, Mi y Sol, en caso de utilizar todas las notas, al haber cuatro voces, una de las tres aparecerá

dos veces.

- Quintas y Octavas consecutivas: no debe haber una distancia de quinta o de octava entre las mismas voces de forma consecutiva.

  Para evitar que esto ocurra, se comprobará la distancia entre cada voz y cuando se detectan las voces en las que aparecen estos intervalos se eliminarán los valores posibles del acorde previo y del posterior, que habiendo ya una nota definitiva, provoquen que la distancia entre las mismas voces sea una quinta o una octava dependiendo de la distancia que se haya detectado.

  Es decir, si en la voz contralto hay un Do y en el soprano un Sol (distancia de 3 tonos y medio sin tener en cuenta la octava a la que pertenece cada uno) lo más conveniente sería que ni en el acorde anterior ni siguiente tuviesen por ejemplo la contralto un Re y la soprano un La.

- Voces cruzadas: Las voces no pueden cruzarse, una voz aguda nunca puede encontrarse por debajo de otra más grave.

  En el sistema se buscarán las notas que ya se han elegido y se compararán con las de la voz superior eliminando los valores que se encuentren por debajo del valor de esa nota y con la voz inferior eliminando los valores que se encuentren por encima. De esta forma se asegura que las voces no se van a cruzar. Solo se compararán las voces inmediatamente superior e inferior porque en caso de que haya alguna nota que se pueda cruzar en otra voz, siempre será eliminada en el momento en que se ponga una nota definitiva en la voz intermedia.

  En este caso, si el tenor tiene un valor de Fa de la octava 4, la contralto no podrá tener un valor de Si de la tercera octava, pero sí de la cuarta.

- El primer y último acorde de la composición debe corresponder al primer grado de la tonalidad en la que está la melodía.

Se eliminarán todas las posibilidades que no pertenezcan al grado I de la tonalidad siempre y cuando la nota de la melodía pueda pertenecer a ese acorde.

- Acordes de grado VII: En la armonía clásica este acorde no se usa para armonizar.

  Como el sistema estará preparado para poderse extender para otros estilos que no se correspondan con la coral clásica, este acorde será contemplado, por tanto se eliminará esa posibilidad para que no pueda ser elegida.

- Distancia entre soprano y contralto: La distancia entre estas voces no debe superar la octava.

  Como la voz de soprano siempre serán notas definitivas, solo hay que eliminar las posibilidades del contralto que haya que superen esta distancia.

  Como ilustración de esta regla se tomará como ejemplo un valor de Fa de la quinta octava en la soprano no podrá haber un Re de la cuarta puesto que la distancia entre ambas es de una doceava.

- Distancia entre contralto y tenor: La distancia entre estas voces no debe superar la octava.

  Para la comprobación de esta regla habrá que tener en cuenta dos casos, que la nota definitiva sea el contralto, en cuyo caso se eliminan todas las posibilidades del tenor que superen la octava, y que la nota definitiva sea el tenor, en este caso las notas posibles que hay que eliminar son las del contralto.

  Este caso es igual que el anterior pero entre distintas voces. Si la contralto tiene un valor de La de la cuarta octava el valor del tenor no podrá ser un Fa de la tercera.

- Diagrama de enlazado de acordes recomendable:

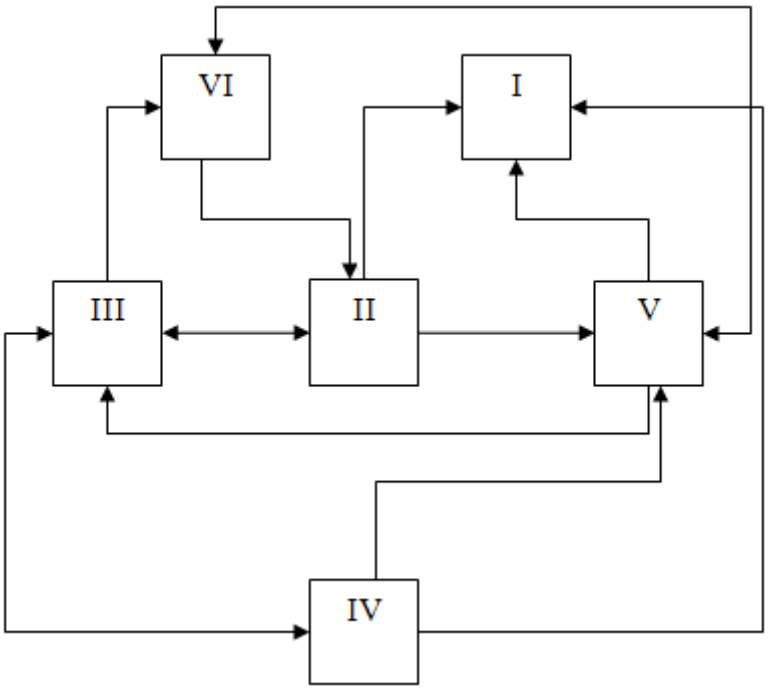

Figura 32. Enlaces posibles entre los grados de las tonalidades

En el diagrama de la Figura 32 se indica la relación que hay entre los acordes en la armonización. Desde el grado I se puede pasar a cualquier acorde por lo que no se dibujan flechas por claridad en el dibujo. Del grado II se puede llegar al grado III, I o V, lo que indica que una posición armonizada con este acorde su enlazado óptimo según este tratado sería a uno de esos acordes. Del grado III se pasa al II, IV o VII. Del IV se procurará que los enlaces vayan al I, al III o al V. En el caso de la armonización con grado V, los enlaces irán a I, III o IV grado. Y los de VI a V o II. El acorde de VII no aparece en el diagrama porque, como se ha indicado con anterioridad, este grado no se usa para la armonización de corales.

En la armonización de una melodía no se busca un acorde distinto cada vez que cambia una nota, sino que sólo se armonizan las partes fuertes del compás, es decir, donde cae el pulso. Con esto se presenta el problema de las figuras que ocupan más de

una parte fuerte como es el caso de las blancas en los compases de 2/4, 3/4 y 4/4. En este caso se armoniza la figura en la primera parte fuerte y se mantiene dicha armonización hasta que cambia la figura en valor fuerte. En el caso de estos compases, una negra con puntillo y una corchea, se armonizaría solo la negra, ya que su valor es superior a una parte fuerte y se calcularía de nuevo el acorde en la figura que prosiga a la corchea.

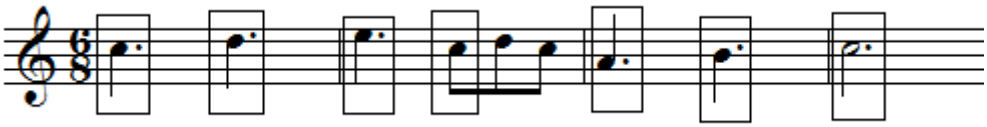

Figura 33. Notas que se armonizan en un compás de 6/8

En la Figura 33 aparecen recuadradas las notas que en un caso de 6/8, siguiendo el criterio elegido, deberán ser armonizadas. En el caso de este compás la unidad de armonización es la negra con puntillo por lo que las negras con puntillo que ocupan una parte múltiplo de la unidad de armonización (en el caso del ejemplo todas), así como también las blancas con puntillo cuyo valor es el doble de la medida de armonización, aparecen recuadradas. Sin embargo, en el caso de las corcheas, al ser su valor menor que el de la unidad de armonización, solo será armonizada la primera de cada grupo.

En la elección del acorde hay que evitar armonizar con alguno que contenga una nota un semitono por debajo de la nota. La forma más segura de hacerlo es emplear acordes que contengan dicha nota, de esta forma se puede estar seguro de que la distancia mínima a las notas que se emplean en el acorde es de cómo mínimo una tercera.

Como las reglas no son unas directrices cerradas existen múltiples soluciones al problema, por lo que la misma melodía se puede armonizar correctamente de más de una forma. Esto implica que una armonización hecha por distintas personas, o por la misma pero tomando distintas decisiones será distinta. La armonía da carácter a la obra, por

tanto estas soluciones sonarán distintas e incluso puede llegar a parecer según la interpretación humana que son completamente diferentes aunque provienen de la misma premisa.

3.2. Diseño del sistema

El objetivo del sistema es crear un acompañamiento armónico para una melodía basándose en unas reglas dadas. Para poder aplicar esas reglas se crean de forma previa todas las posibilidades y se introducen en el motor de reglas que va eliminando posibilidades hasta que, o bien solo queda una y la pasa a definitiva, o encuentra una posibilidad que cumple alguna de las reglas.

La decisión de la armonización está totalmente basada en las reglas, con lo que si las reglas no están bien formadas, o tienen contradicciones de forma que elimine todas las posibilidades introducidas al sistema, no habrá una solución. Por tanto a la hora de diseñar las reglas el usuario tendrá que prestar especial atención a que se cumplan estas condiciones puesto que al estar separada la información de la inteligencia, el sistema solo se preocupa de interpretar la información MIDI tanto de entrada como la que devuelve el bloque de reglas.

Al separar estas dos partes se permite que el sistema sea muy versátil, puesto que simplemente cambiando las reglas aplicadas sin hacer más modificaciones, se consigue un estilo de armonización completamente nuevo. Otra ventaja es que al permitir la tecnología Drools la utilización de varios archivos de reglas de forma simultánea, se pueden hacer pequeñas modificaciones sobre un sistema de reglas ya existente con un archivo adicional que aplique dichos cambios o adiciones sin modificar el estilo original.

3.2.1. Estructura del sistema

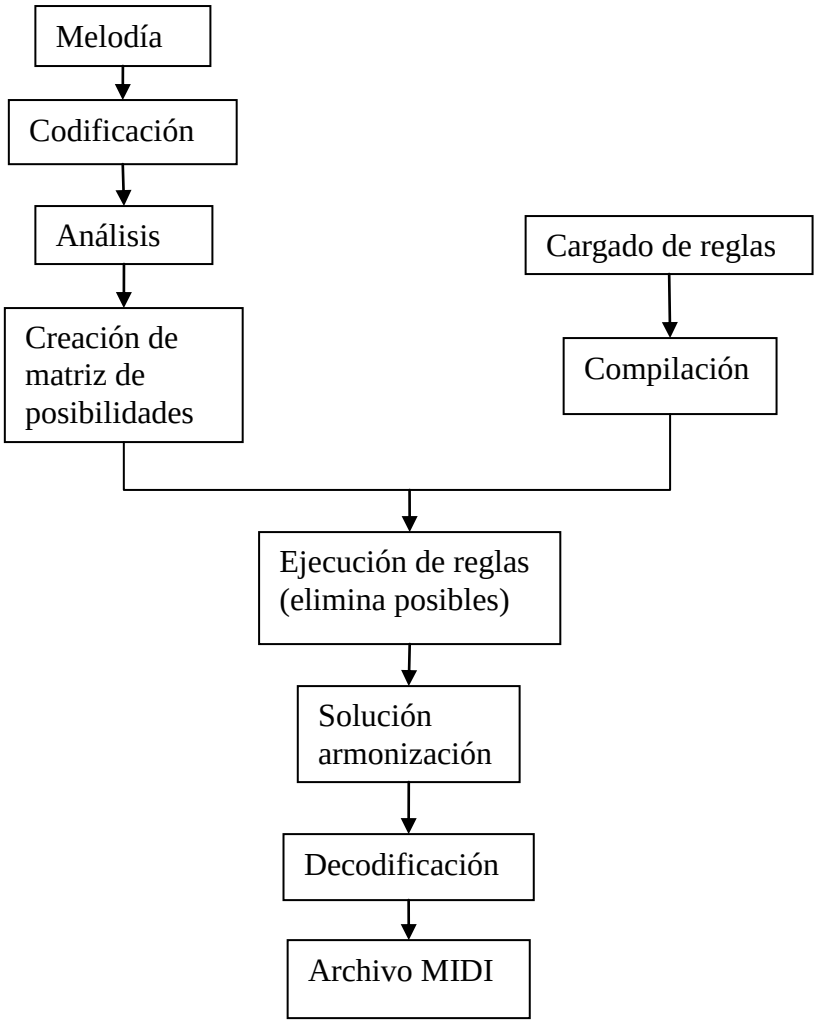

Figura 34. Diagrama de bloques del sistema

Al sistema se le introduce la melodía que se quiere armonizar con información adicional que es la tonalidad y el compás, y el archivo de reglas que determina el estilo de la armonización. La compilación de las reglas se hace mediante métodos proporcionados para drools en Java que se detallarán en puntos posteriores. Sin embargo, la codificación de la melodía sí que hay que hacerla de forma específica puesto hay que adaptar la entrada a los datos del sistema, ya que necesita cosas que MIDI no proporciona como la posición que ocupa en la secuencia a armonizar (que hay que analizar previamente puesto que no todas las notas tienen que armonizarse) y el acorde al que pertenecen. En

el caso de la melodía el acorde al que pertenece se interpreta como uno genérico que se ha llamado 0 para que no condicione los acordes en la decisión. Aquí hay que tener en cuenta que no todas las notas tienen que ser cambiadas de formato sino que solo han de hacerlo las notas que tienen que ser armonizadas según el criterio elegido.

Una vez que se tienen las notas en el formato necesario se analizan las notas de la melodía para saber con qué acordes puede ser armonizada. Esta información se usará en la creación de la matriz de posibilidades. Esta matriz contiene todas las notas posibles que cumplen dos premisas, pertenecen al acorde de un grado específico, y dicho acorde puede usarse en la armonización de la posición. Para ello se hacen las dos comprobaciones y solo se añaden a la matriz que se introducirá en el sistema de reglas aquellas que cumplan esas premisas para cada posición y voz sin tener en cuenta ningún otro factor ya que éstos se aplicarán con las reglas.

Una vez preparados los datos que se van a introducir en el sistema, las notas que no se pueden cambiar (a partir de ahora finales) por pertenecer a la melodía y todas las posibilidades (a partir de ahora posibles) del resto de voces, se procede a la ejecución de las reglas compiladas que se encargarán de encontrar notas finales para cada posición y voz devolviendo una solución que contiene todas esas notas en el orden en el que se han ido decidiendo puesto que lo que se recupera es lo que se ha escrito en la memoria de trabajo.

Debido a que las notas son devueltas en el orden de decisión y no en el orden en el que deben ser escritos, la decodificación constará de dos partes. Primero se ordenará la secuencia de notas según la posición que ocupan en la partitura y posteriormente se traducirá a notas MIDI. En esta traducción hay que tener en cuenta la duración de las notas que se calcula de forma similar a la decisión tomada de que notas se armonizan y cuáles no. Una vez se tiene toda esta información preparada para un archivo MIDI, se crea y se reproduce.

El sistema de composición se puede clasificar en bloques generales que pueden ser cambiados según los criterios que se quieran seguir en la armonización y la obra que se quiera armonizar. Por un lado se encuentran las reglas que se cumplen para llegar a una solución y por otro la melodía que se quiere armonizar junto con todas las posibles soluciones para ella.

3.2.2. Gestión de información

En el sistema se manejan tres tipos de información: la entrada al sistema, la entrada al motor de reglas y la salida del sistema.

Al sistema se le proporciona una información inicial consistente en un archivo MIDI con la melodía que se tiene que armonizar (perteneciente a la voz de la soprano) e información adicional con la tonalidad de la obra. Esta información adicional se utiliza para el cálculo de los acordes que se pueden usar en la armonización ya que cada tonalidad tiene unos acordes distintos. La tonalidad consta de dos parámetros, el nombre y el modo. La información de estos acordes no está previamente almacenada, es decir, no se almacena información para cada tonalidad puesto que habría que tener mucha almacenada, y como los acordes cumplen una relación que depende del modo y del nombre de la tonalidad, se pueden calcular optimizando el volumen que ocupa el programa en memoria. Este cálculo se puede modificar si se quieren emplear otro tipo de acordes como son los de cuatríada.

En cuanto a la entrada al motor de reglas se ha elegido una solución que introduce al sistema todas las posibilidades que posteriormente se van eliminando como se ha indicado en el diagrama de bloques. Para ello, se calcula con respecto a la tonalidad y modo, a que acordes puede pertenecer cada nota y se añaden a la sesión. El número de posibilidades para cada posición puede cambiar debido al número de acordes con el que

se puede armonizar o el número de notas que dentro del rango introducido pertenecen a esos acordes, pero no varía entre voces, ya que en esta parte del sistema no se sabe qué criterios se van a seguir para la armonización.

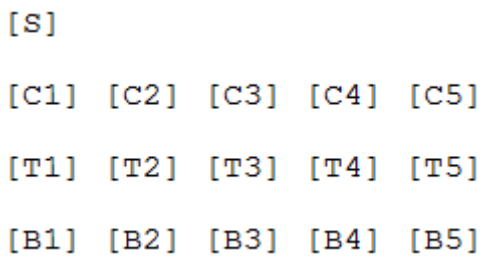

Figura 35. Entrada genérica al sistema

En la Figura 35 se escribe de forma esquemática una posible entrada al sistema. La S representa la voz soprano sobre la que no hay que tomar ninguna decisión porque está considerada nota fina. Las C, T y B son los valores para contralto, tenor y bajo respectivamente y los números representan los valores posibles que pueden tomar siendo iguales en las tres voces, es decir C3, T3 y B3 tienen el mismo valor pero pertenecen a distintas voces.

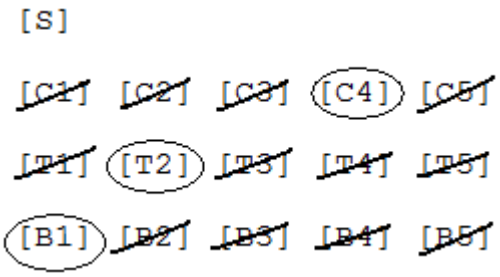

Figura 36. Salida del sistema tras las decisiones

Una vez que se aplican las reglas el sistema pasará a finales una serie de notas descartando el resto. La salida del motor de reglas es una nota para cada voz en cada posición. Las notas de una misma posición pertenecen todas las notas al mismo acorde. Al no encontrarse esta salida en formato MIDI, hay que hacer la conversión al estándar MIDI para que pueda ser reproducido fuera del programa.

El archivo MIDI que se da al usuario como resultado de la ejecución consta de cuatro pistas que contienen cada una de las voces e información sobre el compás en el que está escrito. En la notación de coral clásica se suelen representar generalmente las notas femeninas en un pentagrama y las masculinas en otro, pero para poder recuperar las voces por separado se ha optado por almacenarlas de forma independiente, de esta forma se puede editar de forma cómoda desde programas externos.

3.2.3. Clases Java

En el sistema se tienen cinco clases Java. La clase `Nota` define la estructura de las notas en la forma que el sistema va a manejar la información. Contiene toda la información necesaria y métodos para extraerla. De esta clase extienden dos clases en las que se crean constructores. La razón para tener dos clases es que se ha diseñado un tipo de objetos para la notas que se han elegido como definitivas en el sistema tras la aplicación de reglas o bien porque son la entrada del sistema, y otro tipo para los que aún pueden ser descartados porque no se ha tomado una decisión sobre ellos.

Las otras clases son `Armonizador` que hace las funciones de main del sistema en el que se hacen las transformaciones e interpretaciones para adaptar los datos al bloque de reglas, y `Compositor`, donde se crea el entorno del sistema experto y se ejecutan las reglas.

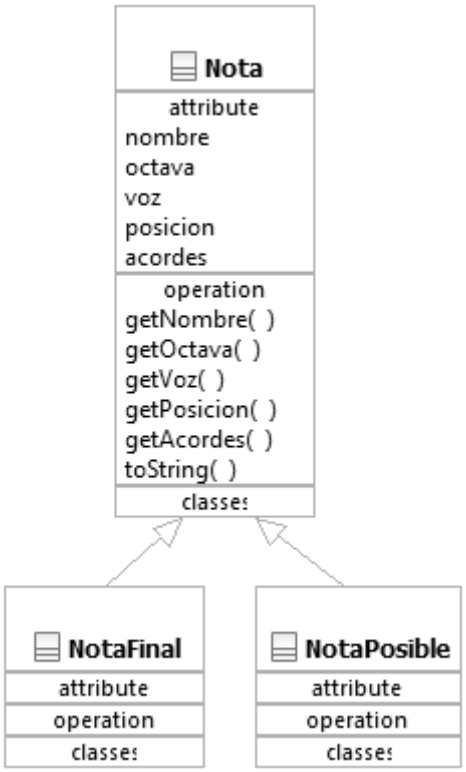

Figura 37. Diagrama de clases Nota, NotaFinal y NotaPosible

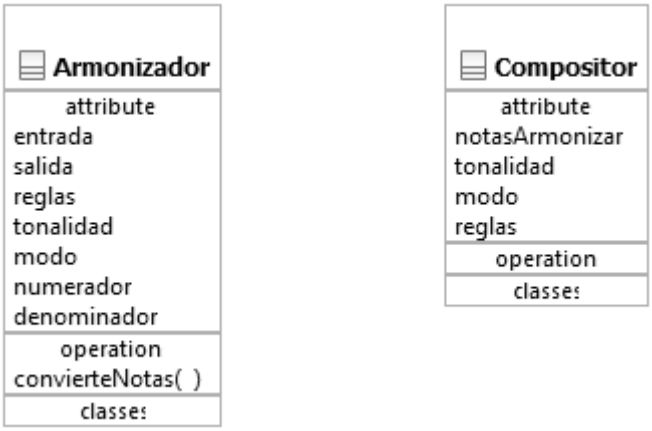

Figura 38. Diagrama de clases Armonizador y Compositor

Clase Nota

La clase `Nota` es una clase abstracta que conforma el esqueleto de los datos que se

introducirán en el sistema experto. Las notas tendrán 5 datos que conformarán la información necesaria de estos objetos para su utilización en las reglas y decisiones. Estos datos son los mostrados en la siguiente tabla:

```
private int nombre;
private int octava;
private int voz;
private int posicion;
private int acordes;
```

Tabla 2. Datos correspondientes a la clase Nota

El `nombre` es un número que representa a un nombre de la escala tonal. Los sonidos que son equivalentes pero con distinto nombre se representan con el mismo número ya que lo que se tiene en cuenta en este sistema son las matemáticas que se pueden aplicar a la música. La `octava` es el número que acompaña al nombre en la notación americana, es decir el C4 tiene como nombre C (Do) y como octava el valor 4. La `voz` es la representación de a cuál de las cuatro voces del coro pertenece. La `posición` es el lugar de las notas que se armonizan ocupa en la partitura. Y los `acordes` el grado del acorde al que pertenece esa nota.

```
public static final int Cb = 11;
public static final int C = 0;
public static final int Cs = 1;
public static final int Db = 1;
public static final int D = 2;
public static final int Ds = 3;
public static final int Eb = 3;
public static final int E = 4;
public static final int F = 5;
public static final int Fs = 6;
public static final int Gb = 6;
public static final int G = 7;
public static final int Gs = 8;
public static final int Ab = 8;
public static final int A = 9;
public static final int As = 10;
public static final int Bb = 10;
public static final int B = 11;
public static final int Bs = 0;
```

Tabla 3. Representación numérica del nombre de las notas

En ese sistema de notación se podrían contemplar los dobles sostenidos y bemoles, pero en el problema al que se busca solución no es habitual encontrarlo ya que es más propio de música para orquesta moderna o como resultado de los trasportes dados por la adaptación de partituras que no están escritas para el instrumento que las toca, por lo que se ha decidido simplificar el sistema no contemplando esta posibilidad.

En la clase `Nota` también se almacena información de a qué acorde de la tonalidad pertenece esa nota. Esta información es útil a la hora de tomar decisiones sobre el enlazado de acordes. También se dispone de información de qué voz se está armonizando con esa nota ya que cada una tiene sus propias notas y eliminar una como posible en una voz no implica que forzosamente esa nota deba ser también descartada para otra.

```
public static final int I = 1;
public static final int II = 2;
public static final int III = 3;
public static final int IV = 4;
public static final int V = 5;
public static final int VI = 6;
public static final int VII = 7;

public static final int SOPRANO = 0;
public static final int CONTRALTO = 1;
public static final int TENOR = 2;
public static final int BAJO = 3;
```

Tabla 4. Representación numérica de los acordes y las voces

En esta clase también existen dos arrays que se usan para poder representar la información de una nota de forma que los usuarios puedan entenderlo. Estos arrays son de nombres de notas en las que el número del nombre se corresponde con la posición del array, por lo que solo se representan como notas naturales o sostenidas y los nombres de las voces de los cantantes.

```
public static final String NOTAS[] = {"C", "C#", "D", "D#", "E", "F","F#",
"G", "G#", "A", "A#", "B"};
```

```
public static final String VOCES[] = {"SOPRANO", "CONTRALTO", "TENOR",
"BAJO"};
```

Tabla 5. Arrays de Strings de la clase Nota

Para esta clase se tienen dos constructores, uno que crea el objeto a partir de todos los datos que necesita un objeto de la clase `Nota` y otro que se crea a partir de otro objeto de esta clase. Este último es necesario porque ésta es una clase desde la que se crean dos clases, `NotaFinal` y `NotaPosible`, siendo NotaPosible la representación de notas que aun no se ha decidido que son las que van a pertenecer a esa posición y `NotaFinal` las que ya son definitivas. Como inicialmente todas las notas pertenecen a la clase `NotaPosible` se crea un mecanismo para que cuando se decida, según las reglas, que una debe ser final se pueda crear utilizando este objeto sin necesidad de extraer todos los atributos.

```
public Nota(int nombre, int octava, int voz, int posicion, int acordes)
{
   this.nombre = nombre;
   this.octava = octava;
   this.voz = voz;
   this.posicion = posicion;
   this.acordes = acordes;
}
```

Figura 39. Constructor a partir de todos los atributos

```
public Nota(Nota valorNota)
{
   this(valorNota.getNombre(), valorNota.getOctava(),
   valorNota.getVoz(), valorNota.getPosicion(),
   valorNota.getAcordes());
 }
```

Figura 40. Constructor a partir de otro objeto Nota

En esta clase también se tienen métodos para poder utilizar la información contenida en las variables del objeto. Estos métodos son `getNombre()` que devuelve el entero correspondiente a la variable nombre, `getOctava()` que devuelve la octava a la que pertenece la nota, `getVoz()` que devuelve la voz a la que está asignada esa nota, `getPosicion()` que devuelve la posición dentro de la composición que tiene asignada esa nota, y `getAcordes()` que devuelve el entero que representa al acorde de tríada.

También se dispone de un método `toString()` que devuelve un String con la información que puede ser entendida por el usuario que se compone de la voz, posición, nombre de la nota en notación americana y el acorde al que pertenece.

```
public String toString()
 {
    return Nota.VOCES[voz] + "_" + posicion + "_" + Nota.NOTAS[nombre] +
    octava + "_" + acordes;
       }
```

Figura 41. Método toString() de la clase Nota

Clases NotaFinal y NotaPosible

Estas clases extienden de la clase `Nota`. La implementación es la misma a excepción del nombre ya que lo que almacenan son objetos `Nota`. La diferencia fundamental es de uso, ya que `NotaPosible` representa los objetos que aún no son una solución sino que son objetos sobre los que aún se tienen que aplicar reglas para decidir si se eliminan o si pasan a ser solución, y `NotaFinal` son objetos que ya se han decidido como solución por lo que no serán cambiados y se usarán para tomar decisiones sobre los que aún son posibles. Una vez que sólo se tienen objetos finales se concluye que se tiene una solución válida al problema.

```
public class NotaFinal extends Nota
{
   public NotaFinal(int nombre, int octava, int voz, int posicion, int
   acordes)
   {
      super(nombre, octava, voz, posicion, acordes);
   }

   public NotaFinal(Nota valorNota)
   {
      super(valorNota);
   }
}
```

Figura 42. Clase NotaFinal que extiende de Nota

```
public final class NotaPosible extends Nota
{
   public NotaPosible(int nombre, int octava, int voz, int posicion, int
acordes)
   {
      super(nombre, octava, voz, posicion, acordes);
   }

   public NotaPosible(Nota valorNota)
   {
      super(valorNota);
   }
}
```

Figura 43. Clase NotaPosible que extiende de Nota

Clase Compositor

La clase `Compositor` es la que se encarga del sistema experto creando todo el entorno necesario para la ejecución de las reglas. La clase recibe varios parámetros cuyos valores posibles se detallan en la siguiente tabla.

Parámetro	notasArmonizar	tonalidad	modo	reglas
Valor	Vector <Nota>	int 0 - 11	char m/M	String ___.drl

Tabla 6. Parámetros de entrada clase Compositor

En Drools se pueden usar dos tipos de sesiones, con estado (`StatefullKnowledgeSession`) y sin estado (`StatelessKnowledgeSession`). En este caso se ha optado por la versión con estado dado que es un problema que hay que abordar de forma global y no es posible obtener un resultado inmediato por cada entrada sino uno global dependiente del resto de entradas. Esto se debe a la dependencia de la armonía de los enlaces.

Al usar la sesión con estado se consigue introducir en el sistema toda la información antes de que se ejecute el archivo de reglas y se haga todo el análisis consecuente de éstas, que son los enlaces y las notas con las que se puede armonizar cada nota que se derivan de ello.

```
    /** La base de conocimiento */
    private KnowledgeBase solverRuleBase;
    final KnowledgeBuilder kbuilder;

    /** La stateful session working memory para la base de conocimiento */
    private StatefulKnowledgeSession    solverStatefulSession;


kbuilder = KnowledgeBuilderFactory.newKnowledgeBuilder();

      kbuilder.add(ResourceFactory.newClassPathResource(reglas,
Compositor.class), ResourceType.DRL);

      if(kbuilder.hasErrors()){
            System.out.println(kbuilder.getErrors().toString());
            throw new RuntimeException("No se puede compilar el archivo de
      reglas");
      }

      final Collection <KnowledgePackage> pkgs =
kbuilder.getKnowledgePackages();

solverRuleBase = KnowledgeBaseFactory.newKnowledgeBase();
solverRuleBase.addKnowledgePackages(pkgs);
solverStatefulSession = solverRuleBase.newStatefulKnowledgeSession();

solverStatefulSession.insert(notasArmonizar.elementAt(posicion));
```

Figura 44. Fragmentos de código con la creación de la StatefullKnowledgeSession

En ese fragmento se observa que para la creación del entorno de ejecución de las reglas, son necesarios también una base de conocimiento (`KnowledgeBase`) y un constructor de conocimiento (`KnowledgeBuilder`).

El `KnowledBase` es un repositorio donde se encuentran todas las definiciones de conocimiento de la aplicación. Éste puede contener reglas, procesos, funciones y tipos de modelos. La base de conocimiento no contiene instancias de datos, también conocidos

como hechos, en cambio las sesiones se crean de esta base de conocimiento y en ellas si se pueden insertar datos y generar hechos.

La sesión además cuenta con una memoria de trabajo que dispone de tres métodos. El método `insert` permite informar a la memoria de trabajo de un dato. Cuando un dato es insertado se examina para hacerlo coincidir con las reglas. Esto quiere decir que todo el trabajo sobre la decisión de activar o no una regla se hace durante la inserción, aunque no es ejecutada hasta que se llama a `fireAllRules()`, lo que se hace una vez que se han acabado de insertar todos los datos.

Otro de los métodos proporcionados es `retract`, que permite eliminar datos de la memoria de trabajo. Lo que implica que no se seguirá comprobando en las reglas y que las activaciones de reglas que derivan de él son canceladas.

Por último se dispone también del método `update`. Con este se notifica la modificación de un dato para que pueda ser procesado. Internamente una modificación se considera un `retract` seguido de un `insert`, lo que quiere decir que el motor elimina un dato de la memoria de trabajo y posteriormente lo vuelve a insertar. Se usa en los casos en los que se necesita notificar a la memoria de trabajo de un cambio en un objeto que no puede notificar por si mismo ese cambio.

Por tanto, lo primero que se hace en esta clase es crear el entorno de drools para poder ejecutar el sistema experto. Para ello se crea un `knowledgeBuilder` descrito anteriormente al que se añade una fuente classPath del tipo DRL, es decir, se le agrega el archivo de reglas y se comprueba si tiene errores. A continuación se extraen las reglas del archivo y se crea la sesión con la que se trabajará.

```
        kbuilder = KnowledgeBuilderFactory.newKnowledgeBuilder();
        kbuilder.add(ResourceFactory.newClassPathResource(reglas,
Compositor.class), ResourceType.DRL);

        if(kbuilder.hasErrors()){
```

```
            System.out.println(kbuilder.getErrors().toString());
            throw new RuntimeException("No se puede compilar el archivo de
reglas");
      }

      final Collection <KnowledgePackage> pkgs =
kbuilder.getKnowledgePackages();

      solverRuleBase = KnowledgeBaseFactory.newKnowledgeBase();
      solverRuleBase.addKnowledgePackages(pkgs);
      solverStatefulSession =
solverRuleBase.newStatefulKnowledgeSession();
```

Figura 45. Inicialización del entorno Drools

Una vez que se tiene todo el entorno preparado, el sistema analiza la melodía para saber con qué acordes se puede armonizar. Para ello se recorre el vector de notas añadiendo a un vector auxiliar la información de con qué acordes se puede armonizar dependiendo del modo. Previamente añade a la sesión las notas que se han de armonizar.

Con la relación numérica que existe en los acordes de las tonalidades se comprueba a qué acordes puede pertenecer esa nota, ya que la armonización se puede hacer solo de los acordes que la incluyan. Para ello se compara el valor del nombre de la nota con los tres valores de las notas de cada acorde para ver si se encuentra entre ellas. Si esta comparación tiene un resultado afirmativo se añadirá al vector auxiliar el valor del acorde para que posterior mente se puedan agregar todos los valores pertenecientes a estos acordes como notas posibles.

```
      for(int posicion=0; posicion<notasArmonizar.size(); posicion++) {

            /* Melodía que hay que armonizar */
            solverStatefulSession.insert(notasArmonizar.get(posicion));

            Nota aux = notasArmonizar.get(posicion);
            Vector<Integer> acordesPosibles = new Vector<Integer>();

      /* Comprobamos a que acordes puede pertenecer la nota del soprano*/
            if(modo=='M') {
              if(aux.getNombre()==tonalidad ||
aux.getNombre()==(tonalidad+4)%12 || aux.getNombre()==(tonalidad+7)%12)
                  acordesPosibles.add(Nota.I);
              if(aux.getNombre()==(tonalidad+2)%12 ||
aux.getNombre()==(tonalidad+5)%12 || aux.getNombre()==(tonalidad+9)%12)
```

```
                    acordesPosibles.add(Nota.II);
                if(aux.getNombre()==(tonalidad+4)%12 ||
aux.getNombre()==(tonalidad+7)%12 || aux.getNombre()==(tonalidad+11)%12)
                    acordesPosibles.add(Nota.III);
                if(aux.getNombre()==(tonalidad+5)%12 ||
aux.getNombre()==(tonalidad+9)%12 || aux.getNombre()==tonalidad)
                    acordesPosibles.add(Nota.IV);
                if(aux.getNombre()==(tonalidad+7)%12 ||
aux.getNombre()==(tonalidad+11)%12 || aux.getNombre()==(tonalidad+2)%12)
                    acordesPosibles.add(Nota.V);
                if(aux.getNombre()==(tonalidad+9)%12 ||
aux.getNombre()==tonalidad || aux.getNombre()==(tonalidad+4)%12)
                    acordesPosibles.add(Nota.VI);
                if(aux.getNombre()==(tonalidad+11)%12 ||
aux.getNombre()==(tonalidad+2)%12 || aux.getNombre()==(tonalidad+5)%12)
                    acordesPosibles.add(Nota.VII);

            }
        else if(modo=='m')
        {
                if(aux.getNombre()==tonalidad ||
aux.getNombre()==(tonalidad+3)%12 || aux.getNombre()==(tonalidad+7)%12)
                    acordesPosibles.add(Nota.I);
                if(aux.getNombre()==(tonalidad+2)%12 ||
aux.getNombre()==(tonalidad+5)%12 || aux.getNombre()==(tonalidad+8)%12)
                    acordesPosibles.add(Nota.II);
                if(aux.getNombre()==(tonalidad+3)%12 ||
aux.getNombre()==(tonalidad+7)%12 || aux.getNombre()==(tonalidad+10)%12)
                    acordesPosibles.add(Nota.III);
                if(aux.getNombre()==(tonalidad+5)%12 ||
aux.getNombre()==(tonalidad+8)%12 || aux.getNombre()==tonalidad)
                    acordesPosibles.add(Nota.IV);
                if(aux.getNombre()==(tonalidad+7)%12 ||
aux.getNombre()==(tonalidad+10)%12 || aux.getNombre()==(tonalidad+2)%12)
                    acordesPosibles.add(Nota.V);
                if(aux.getNombre()==(tonalidad+8)%12 ||
aux.getNombre()==tonalidad || aux.getNombre()==(tonalidad+3)%12)
                    acordesPosibles.add(Nota.VI);
                if(aux.getNombre()==(tonalidad+10)%12 ||
aux.getNombre()==(tonalidad+2)%12 || aux.getNombre()==(tonalidad+5)%12)
                        acordesPosibles.add(Nota.VII);
            }
```

Figura 46. Análisis de acordes para armonizar la melodía

No hay que olvidar que la escala natural de una tonalidad se forma con distintas relaciones según el modo y por tanto también los acordes de tríada serán distintos. Es por esto por lo que se han de hacer comparaciones distintas según el modo en el que se esté armonizando. Como se aprecia en las siguientes figuras la tónica de la escala es la

misma, pero aparecen alteraciones en las notas que modifican la distancia entre ellas.

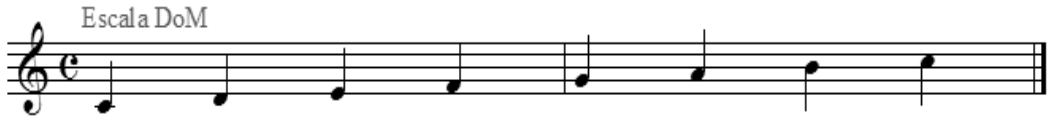

Figura 47. Escala de Do Mayor

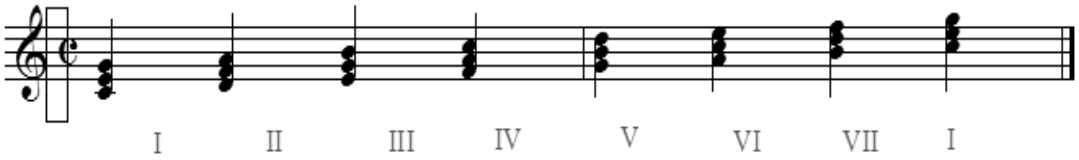

Figura 48. Grados de Do Mayor

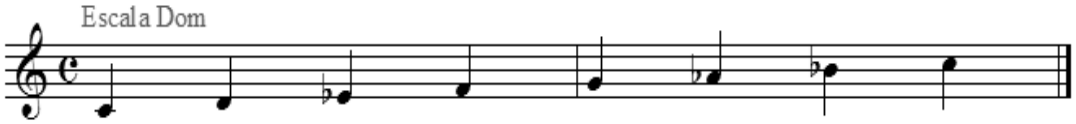

Figura 49. Escala de Do menor

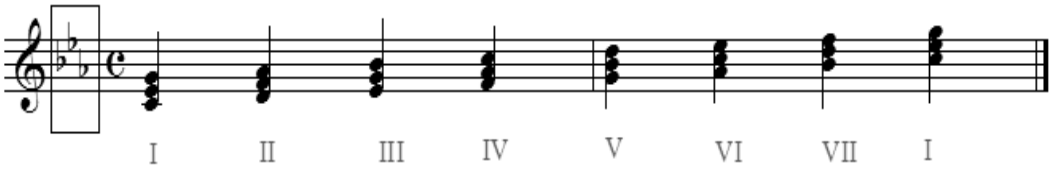

Figura 50. Grados de Do menor

Una vez que se ha llevado a cabo el análisis y almacenado en un vector auxiliar se procede, de forma similar, a insertar como valores posibles todas las notas que pueden pertenecer a cada posición de la composición. Para ello además del análisis de cada nota para ver a qué acorde puede pertenecer se compara también que la nota a armonizar tenga como posible dicho acorde. Esto se lleva a cabo dentro de un bucle para hacerlo para todas las voces.

```
for(int octava=2; octava<=5; octava++) {
  for(int nota=Nota.C; nota<=Nota.Cb; nota++) {
    if(modo=='M') {
```

```
            if((nota==tonalidad || nota==(tonalidad+4)%12 ||
      nota==(tonalidad+7)%12) && acordesPosibles.contains(Nota.I))
            solverStatefulSession.insert(new NotaPosible(nota, octava,
      voz, posicion, Nota.I));
            if((nota==(tonalidad+2)%12 || nota==(tonalidad+5)%12 ||
      nota==(tonalidad+9)%12) && acordesPosibles.contains(Nota.II))
            solverStatefulSession.insert(new NotaPosible(nota, octava,
      voz, posicion, Nota.II));
            if((nota==(tonalidad+4)%12 || nota==(tonalidad+7)%12 ||
      nota==(tonalidad+11)%12) && acordesPosibles.contains(Nota.III))
            solverStatefulSession.insert(new NotaPosible(nota, octava,
      voz, posicion, Nota.III));
            if((nota==(tonalidad+5)%12 || nota==(tonalidad+9)%12 ||
      nota==tonalidad) && acordesPosibles.contains(Nota.IV))
            solverStatefulSession.insert(new NotaPosible(nota, octava,
      voz, posicion, Nota.IV));
            if((nota==(tonalidad+7)%12 || nota==(tonalidad+11)%12 ||
      nota==(tonalidad+2)%12) && acordesPosibles.contains(Nota.V))
            solverStatefulSession.insert(new NotaPosible(nota, octava,
      voz, posicion, Nota.V));
            if((nota==(tonalidad+9)%12 || nota==tonalidad ||
      nota==(tonalidad+4)%12) && acordesPosibles.contains(Nota.VI))
            solverStatefulSession.insert(new NotaPosible(nota, octava,
      voz, posicion, Nota.VI));
            if((nota==(tonalidad+11)%12 || nota==(tonalidad+2)%12 ||
      nota==(tonalidad+5)%12) && acordesPosibles.contains(Nota.VII))
            solverStatefulSession.insert(new NotaPosible(nota, octava,
      voz, posicion, Nota.VII));
      }
    if(modo=='m'){
            if((nota==tonalidad || nota==(tonalidad+3)%12 ||
      nota==(tonalidad+7)%12) && acordesPosibles.contains(Nota.I))
            solverStatefulSession.insert(new NotaPosible(nota, octava,
      voz, posicion, Nota.I));
            if((nota==(tonalidad+2)%12 || nota==(tonalidad+5)%12 ||
      nota==(tonalidad+8)%12) && acordesPosibles.contains(Nota.II))
            solverStatefulSession.insert(new NotaPosible(nota, octava,
      voz, posicion, Nota.II));
            if((nota==(tonalidad+3)%12 || nota==(tonalidad+7)%12 ||
      nota==(tonalidad+10)%12) && acordesPosibles.contains(Nota.III))
            solverStatefulSession.insert(new NotaPosible(nota, octava,
      voz, posicion, Nota.III));
            if((nota==(tonalidad+5)%12 || nota==(tonalidad+8)%12 ||
      nota==tonalidad) && acordesPosibles.contains(Nota.IV))
            solverStatefulSession.insert(new NotaPosible(nota, octava,
      voz, posicion, Nota.IV));
            if((nota==(tonalidad+7)%12 || nota==(tonalidad+10)%12 ||
      nota==(tonalidad+2)%12) && acordesPosibles.contains(Nota.V))
            solverStatefulSession.insert(new NotaPosible(nota, octava,
      voz, posicion, Nota.V));
            if((nota==(tonalidad+8)%12 || nota==tonalidad ||
      nota==(tonalidad+3)%12) && acordesPosibles.contains(Nota.VI))
            solverStatefulSession.insert(new NotaPosible(nota, octava,
      voz, posicion, Nota.VI));
            if((nota==(tonalidad+10)%12 || nota==(tonalidad+2)%12 ||
      nota==(tonalidad+5)%12) && acordesPosibles.contains(Nota.VII))
```

```
                solverStatefulSession.insert(new NotaPosible(nota, octava,
            voz, posicion, Nota.VII));
            }
        }
    }
```

Figura 51. Inserción de notas posibles en la sesión

Tras esto se invoca el método `fireAllRules()` de la sesión para arrancar la ejecución de las reglas y obtener un resultado que se recoge, una vez que ha terminado, de la misma sesión, ya que después de la eliminación de las notas posibles consideradas no válidas sólo quedan las que son consideradas solución (notas finales). Una vez obtenidos los objetos se separan en vectores según la voz a la que pertenezcan para poder ser recuperados desde fuera de la clase.

Clase Armonizador

Esta clase hace la función de main de la aplicación. Ésta es la que ejecuta el usuario. Para llevar a cabo su ejecución hay que introducir varios parámetros que se detallan en la siguiente tabla.

Parámetro	Entrada	Salida	reglas	tonalidad	Numerador compás	Denominador compás	modo
Valor	String ___.mid	String ___.mid	String ___.drl	Int 0 -11	int	int	char m/M

Tabla 7. Parámetros de entrada clase Armonizador

Para la lectura y escritura de los archivos MIDI se utilizan librerías de JMusic, que permiten el manejo del contenido de un archivo MIDI, así como su lectura, escritura y reproducción.

```
import jm.JMC;
import jm.music.data.Note;
import jm.music.data.Part;
import jm.music.data.Phrase;
import jm.music.data.Score;
import jm.util.Play;
import jm.util.Read;
import jm.util.Write;
```

Tabla 8. Librerías de jmusic usadas en la clase

En primer lugar se lee el archivo MIDI que se ha indicado como entrada al sistema. Para ello se crea un objeto de la clase Score donde se almacenan los datos del MIDI. También se crea otro para almacenar la salida del sistema, pero no se usará hasta que no se disponga de un resultado del sistema.

```
Score theScore = new Score("input score");
Score nuevoScore = new Score("output score");

String archivo = args[0];
Read.midi(theScore, archivo);

//Aunque solo tendrá una parte hay que cogerla para poder manejarla
Part sopranoIni = theScore.getPart(0);
//inicialmente suponemos que solo va a haber una frase
Phrase fraseSoprano = sopranoIni.getPhrase(0);
```

Figura 52. Obtención de los datos del archivo MIDI

La encapsulación de los datos MIDI hace que haya que hacer varios pasos intermedios para poder obtener las notas. El conjunto de todos los datos se llama `Score`, este a su vez contiene cada voz guardada en objetos llamados `Part`, y cada `Part` contiene `Phrases` que son unidades en las que se pueden meter notas.

Según se comentó en el apartado 3.1, en una armonización no todas las notas se armonizan, sólo las que coinciden con el pulso, es decir si en un pulso hay dos corcheas sólo se armoniza la primera. Esto no es una norma, si no que queda a decisión del compositor ya que se podría elegir armonizar la segunda, si a una corchea le sigue una negra se puede optar por armonizar la negra… La decisión que se ha tomado para este sistema es que se armonizan las notas que comienzan con un pulso, es decir si hay una

negra que no empieza con el pulso no se armonizará. Para ello se calcula un factor con el que se calculará qué notas han de ser armonizadas. En caso de numeradores distintos, como sería un compás medido en semicorcheas (denominador 16) no estaría contemplado, y en caso de compases irregulares (por ejemplo 5/8), dependiendo de cómo estén distribuidos, ésta podría no ser la mejor solución, por lo que tampoco se contempla este escenario. Pero estos casos son muy poco habituales por lo que no habría una mejora significativa en caso de ser incluidos y complicaría la legibilidad del código.

```
if (denominador == 4){
      finCompas = numerador;
      armoniza = 1;
}else if(denominador == 8){
      finCompas = numerador/2;
      armoniza = 1.5;
}else if(denominador == 2){
      finCompas = numerador*2;
      armoniza = 2;
}else
      throw new Exception("Este compás no puede ser armonizado por
este sistema");
```

Figura 53. Cálculo según el compás de que notas hay que armonizar

Una vez que se tienen las notas en formato MIDI hay que pasarlas al formato que se ha definido para el sistema en la clase `Nota`. Para calcular el nombre se calcula el resto de dividir el pitch que se obtiene del objeto `Note` extraído de MIDI por 12 ya que la nota Do siempre se encuentra en múltiplos de este número y cada semitono se encuentra un entero por encima del anterior. Por tanto, si se toma como ejemplo la nota Re que en MIDI toma los valores 2, 14, 26, 38, 50, etc. Se puede ver que en todos los casos el resto de la división por 12 da como resultado 2 que se corresponde con el valor que toma en la clase nota.

La octava se puede obtener cogiendo solo la parte entera de dividir el valor de la nota de MIDI entre 12 y restándole 2. Siguiendo con el ejemplo anterior la parte entera de dividir 2/12 es 0 y en MIDI esta octava se corresponde con la que tiene valor -2 con lo que aplicándole la corrección quedaría con el mismo valor, y de igual manera ocurre con 14/12 cuya parte entera es 1 y tiene la misma correlación con MIDI.

Como la entrada que se está parametrizando es la parte de la soprano, se da un valor 0 a los acordes para no tenerlo en cuenta a la hora de armonizar el resto puesto que no se quiere condicionar al sistema en la resolución, sino que sea él mismo el que tome las decisiones del análisis y la armonización siguiendo las reglas dadas.

```
Vector<Object> notasOriginal = fraseSoprano.getNoteList();
for (int i = 0; i < notasOriginal.size(); i++){
    if (contador == finCompas)
          contador = 0;
    Note notaAux = (Note)notasOriginal.get(i);
    int nombre = notaAux.getPitch()%12-2;
    int octava = (int)notaAux.getPitch()/12;
    int acordes =0;
    NotaFinal nota = new NotaFinal(nombre, octava, Nota.SOPRANO, i,
    acordes);
    double duracion = notaAux.getRhythmValue();
```

Figura 54. Cálculo de los valores de las variables de la clase Nota para notas MIDI

Durante las pruebas se ha observado que la duración de las notas no se devuelve en los números que se esperan que son 0.5 (el valor obtenido oscila entre 0.45 y 0.55) para las corcheas, 1 (el valor obtenido oscila entre 0.95 y 1.05) para las negras, 1.5 para las negras con puntillo (el valor obtenido oscila entre 1.45 y 1.55)… sino que estos valores varían. El margen de variación no supera el 0.05, por lo que se ha creado una parte de código para adecuar las medidas a las esperadas en el sistema. Es necesario que coincidan con las esperadas por cómo se decide qué notas se armonizan y cuáles no. Si no se corrige esta variación podría darse el caso de que no se armonizara ninguna nota ya que al sumar los valores de la duración de las notas podría ser que ninguna coincidiera con la armonización. Por ejemplo si la primera nota de la composición es una negra y su valor obtenido es 1.05, si el resto de las notas coinciden con el valor esperado sólo se armonizará en toda la obra esta nota porque nunca la suma será múltiplo de uno en un caso de denominador 4.

```
if(duracion>0.2 && duracion<0.3)
      notaAux.setRhythmValue(0.25); //semicorchea
else if(duracion>0.45 && duracion<0.55)
      notaAux.setRhythmValue(0.5); //corchea
else if (duracion>0.7 && duracion<0.8)
      notaAux.setRhythmValue(0.75); //corchea con puntillo
else if (duracion>0.95 && duracion<1.05)
      notaAux.setRhythmValue(1); //negra
else if (duracion>1.45 && duracion<1.55)
```

```
      notaAux.setRhythmValue(1.5); //negra con puntillo
else if (duracion>1.95 && duracion<1.05)
      notaAux.setRhythmValue(2); //blanca
else if (duracion>2.95 && duracion<3.05)
      notaAux.setRhythmValue(3); //blanca con puntillo
else if (duracion>3.95 && duracion>4.05)
      notaAux.setRhythmValue(4); //redonda
```

Figura 55. Adaptación del valor de la duración de las notas

Como se observa en el código, la adaptación se hace para los valores comprendidos entre semicorcheas y redondas. Aunque existen valores tanto por encima como por debajo no se utilizan, al menos de forma habitual, en la coral, debido a la complejidad que esto supondría para el canto, por lo que en la aplicación se ha optado por acotar dichos valores a los más habituales.

Una vez que se han adaptado tanto los valores del ritmo como las notas MIDI al formato del sistema, se ha de decidir qué notas se han de armonizar. Para ello se dispone de un factor de armonización llamado `armoniza` cuya función es indicar qué medida mínima es la que se armoniza, es decir, si coincide con que la figura empieza en un múltiplo de ese factor se armoniza, pero si no empieza en esa posición del compás no. Lo que se hace es que se mantiene la armonización de la posición anterior puesto que es el criterio que se ha elegido para esto.

```
if (contador == 0.0 || contador%armoniza == 0.0){
      notasArmonizar.add(nota);
      contador = contador + notaAux.getDuration();
} else{
      contador = contador + notaAux.getDuration();
}
```

Figura 56. Decisión de que notas se armonizan

Una vez adaptado todo al formato que se necesita se crea un objeto de la clase `Compositor` que se encarga de hacer la armonización de la melodía como anteriormente se ha explicado. Para ello se le pasa el vector de reglas adecuado a la clase `Nota`, el

nombre de la tonalidad, el modo y el archivo de reglas que se quiere usar.

```
Compositor composicion = new Compositor (notasArmonizar, tonalidad,
modo, args[2]);
```

Figura 57. Creación del objeto de la clase Composición

Una vez que se han ejecutado las reglas se puede acceder a los vectores que contienen el resultado de cada voz de la clase `Compositor`, puesto que están declarados como `public`. Antes de añadir la información recogida del compositor se calcula la duración de cada nota, ya que no se dispone de esta información en la clase `Nota` por no ser necesaria para la armonización.

El resultado que se obtiene del sistema no tiene el mismo orden que el que se usó de entrada pues se recupera en el orden en el que ha sido insertado en la memoria de trabajo. Por tanto se tiene que ordenar para poder ser guardado en un archivo MIDI.

```
Vector<Nota>vContraltoResultado = composicion.contralto;
Vector<Nota>vTenorResultado = composicion.tenor;
Vector<Nota>vBajoResultado = composicion.bajo;

//Se ordenan las notas para poderlas escribir en orden en el MIDI

Nota[] vContraltoAux = new Nota[vContraltoResultado.size()];
Nota[] vTenorAux = new Nota[vTenorResultado.size()];
Nota[] vBajoAux = new Nota[vBajoResultado.size()];

for(int i=0; i<vContraltoResultado.size(); i++){
      Nota notaAux =  vContraltoResultado.elementAt(i);
      int g = notaAux.getPosicion();
      vContraltoAux[g]=notaAux;
}

for(int i=0; i<vTenorResultado.size(); i++){
      Nota notaAux = vTenorResultado.elementAt(i);
      int g = notaAux.getPosicion();
      vTenorAux[g]=notaAux;
}

for(int i=0; i<vBajoResultado.size(); i++){
      Nota notaAux = vBajoResultado.elementAt(i);
      int g = notaAux.getPosicion();
      vBajoAux[g]=notaAux;
}

Vector<Nota> vContralto = new Vector<Nota>();
```

```
Vector<Nota> vTenor = new Vector<Nota>();
Vector<Nota> vBajo = new Vector<Nota>();

for(int i=0; i<vContraltoAux.length; i++){
      vContralto.add(vContraltoAux[i]);
      vTenor.add(vTenorAux[i]);
      vBajo.add(vBajoAux[i]);
}
```

Figura 58. Ordenación del resultado

Para esta reordenación en primer lugar se recuperan las notas almacenadas para cada voz. Estas notas contienen información sobre la posición de la partitura a la que pertenecen. Una vez se tiene la información, se ordena en unos arrays de objeto `Nota` de forma que cada nota pasa a ocupar la posición correspondiente a su posición en la partitura para introducirlo en el orden correcto en el vector que se usará para crear el archivo MIDI.

Una vez que se sabe el valor para cada nota se crean los elementos necesarios para crear el archivo MIDI nuevo al que hay que añadirle además de la voz original las tres voces obtenidas. Los elementos son un objeto `Part` y otro `Phrase` para cada voz. Al objeto `Phrase` se le añadirán las notas y una vez hecho esto se agregarán las `Phrase` a las `Part` que a su vez se añadirán al `Score` creado al principio de la clase como salida.

```
for (int b=0; b<vContralto.size(); b++){
      fraseSoprano2.add(notasSoprano.elementAt(b));

      Note notaContralto = convierteNotas(vContralto.elementAt(b),
ritmos.elementAt(b));
      fraseContralto.add(notaContralto);

      Note notaTenor =
convierteNotas(vTenor.elementAt(b),ritmos.elementAt(b));
      fraseTenor.add(notaTenor);

      Note notaBajo = convierteNotas(vBajo.elementAt(b),
ritmos.elementAt(b));
      fraseBajo.add(notaBajo);
}

soprano.add(fraseSoprano2);
contralto.addPhrase(fraseContralto);
tenor.addPhrase(fraseTenor);
bajo.addPhrase(fraseBajo);
```

```
nuevoScore.setDenominator(denominador);
nuevoScore.setNumerator(numerador);
nuevoScore.add(soprano);
nuevoScore.add(contralto);
nuevoScore.add(tenor);
nuevoScore.add(bajo);

Write.midi(nuevoScore, args[1]);
```

Figura 59. Escritura de archivo MIDI a partir de la información devuelta por Compositor

Como se aprecia en el código, a un archivo MIDI hay que darle información adicional. Esta información no es obligatoria para que se pueda reproducir, pero como en nuestro caso queremos que se pueda además de reproducir, representar en forma de partitura en programas externos, además de las notas le damos información del compás en el que está escrito. Otra información que se le puede dar es la velocidad a la que puede ser reproducida.

Como se ha indicado antes, el formato de las notas que se usa en el sistema experto no coincide con el formato usado por el estándar MIDI, por lo que se requiere de un método que hace el cálculo inverso al que se hizo para adaptar la entrada en MIDI al sistema. Para ello se ha creado el método `convierteNotas` que recibe una nota en formato `Nota` y la duración y la devuelve en formato `Note`, que es la que se usa para escribir en estándar MIDI.

```
public static Note convierteNotas(Nota nota, double duracion){
  int pitch = (nota.getOctava()*12+2) + nota.getNombre();
  Note midiNote = new Note(pitch, duracion);
  return midiNote;
}
```

Figura 60. Conversión de Nota a Note

En este método se calcula el pitch de la nota a partir de la información de octava y del nombre que contiene el objeto `Nota` y se agrega a un objeto `Note` que es el que se devuelve. Además del pitch se incluye la duración de la nota para que el objeto devuelto también lo tenga.

4 IMPLEMENTACIÓN DE REGLAS

Para la validación de los desarrollos se han implementado dos archivos de reglas. Uno de ellos basado en las reglas de la coral clásica que se explicaron en el apartado 3.1, y otro que deja que el sistema elija el resultado sin restricciones dentro de las posibilidades de cada voz.

4.1. CoralClasica

En este archivo se encuentran las reglas correspondientes a la armonización llevada a cabo según los cánones de la coral clásica. Estas normas son las que tradicionalmente se estudian en ámbitos educativos.

En primer lugar se acotan las voces a las que se va a buscar solución, ya que a la hora de introducir las notas posibles en el sistema no se tiene en cuenta a qué voz pertenecen. En coral, la voz de la contralto se encuentra entre B3 y B4 por lo que las octavas que se encuentren por encima de 4 y por debajo de 3 deben ser eliminadas, así como las notas que perteneciendo a esas octavas se encuentren por encima del Si en la octava 4 y por debajo del Si en la octava 3. La tesitura del tenor se encuentra entre E3 y F4 por lo que todas las notas que se encuentren por debajo del Mi de la tercera octava y por encima del Fa de la cuarta deben ser suprimidos de las notas con las que se puede armonizar esa voz. Por último se tiene la voz del bajo que se comprende entre G2 y A3 por lo que todas las notas que se encuentren fuera de este intervalo tienen que ser suprimidas.

```
rule "Tesitura voz contralto"
      salience 100
      when
            $posible : NotaPosible(voz==Nota.CONTRALTO, octava>4 ||
      (octava==4 && nombre>Nota.B) || (octava==3 && nombre<Nota.B ||
      octava<3))
```

```
	then
		update($posible);
		retract($posible);
		System.out.println("TesCont Se ha eliminado la nota posible "
	+ $posible.toString());
end

rule "Tesitura voz tenor"
	salience 100
	when
		$posible : NotaPosible(voz==Nota.TENOR, (octava>4) ||
	(octava==4 && nombre>Nota.F) || (octava<3) || (octava==3 &&
	nombre<Nota.E))
	then
		retract($posible);
		System.out.println("TesTen Se ha eliminado la nota posible " +
$posible.toString());
end

rule "Tesitura voz bajo"
	salience 100
	when
		$posible : NotaPosible(voz==Nota.BAJO, octava>3 || (octava==3
	&& nombre>Nota.A) || octava<2 || (octava==2 && nombre<Nota.G))
	then
		retract($posible);
		System.out.println("TesBaj Se ha eliminado la nota posible " +
	$posible.toString());
end
```

Figura 61. Acotación de las voces contralto, tenor y bajo

En las reglas se buscan las notas que perteneciendo a una voz se encuentran fuera de su rango permitido y se eliminan del sistema mediante el método retract. Después de eso se imprime por pantalla la nota que se ha eliminado y la regla que se ha cumplido para llevar a cabo esta acción. Estas reglas tienen prioridad (salience) 100 porque son las primeras que se deben ejecutar, ya que de otro modo se podría decidir una nota final que se encontrara fuera del rango permitido.

A continuación, y con una prioridad también alta (90), se tienen varias reglas, que aunque no se corresponden directamente con reglas de armonía, son necesarias para poder llegar a una solución. Una de ellas es eliminar todas las notas posibles cuando se ha decidido una solución válida para una posición de una voz. Esta regla existe y tiene una prioridad alta porque el sistema debe devolver únicamente notas finales (objetos

pertenecientes a la clase NotaFinal) y solo puede haber una nota final para cada posición de cada voz y si esta regla no tuviese una prioridad mayor a las reglas que se obtienen directamente de las de armonía se podría asignar más de una nota final para una voz y posición debido a que el problema no tiene una solución cerrada.

```
rule "#01 Una vez se resuelve una nota se eliminan los valores posibles de
esa posicion"
	salience 90
	when
		$resuelto : NotaFinal($vozResuelta : voz, $posicionResuelta :
	posicion)
		$posible : NotaPosible(voz==$vozResuelta,
	posicion==$posicionResuelta)
	then
		retract($posible);
		update($resuelto);
		System.out.println("01 Se ha eliminado la nota" +
	$posible.toString());
end
```

Figura 62. Regla que elimina los valores posibles cuando existe un valor final

Otra regla del mismo tipo que la anterior es la número 2, que en caso de haber eliminado todos los valores posibles menos uno, cambia el valor a final. En este caso la prioridad también es superior a la mayoría de las reglas de armonía ya que si fuese menor o igual se podría dar el caso que se activase una regla para otra posición que obligase a eliminar ese valor y no se daría solución a esa posición con lo que el resultado final no sería válido.

```
rule "#02 Cuando solo existe una nota posible se pasa a resuelta y se
elimina como valor posible"
	salience 90
	when
		$posible : NotaPosible($vozPosible : voz, $posicionPosible :
	posicion, $nombrePosible : nombre, $octavaPosible : octava,
	$acordesPosible : acordes)
		not (NotaPosible(voz==$vozPosible, posicion==$posicionPosible,
	octava!=$octavaPosible || nombre!=$nombrePosible ||
	(nombre==$nombrePosible && octava==$octavaPosible &&
	acordes!=$acordesPosible)))
	then
		update($posible);
		retract($posible);
		insert(new NotaFinal($posible));
		System.out.println("02 Se ha resuelto la nota " +
	$posible.toString());
```

```
end
```

Figura 63. Regla que cuando solo existe una nota posible la pasa a resuelta

Para evitar elegir como finales acordes que no aparezcan en alguna de las otras voces y al eliminar el resto de acordes esa posición quede sin solución, se eliminarán como notas posibles las pertenecientes a un acorde que no aparezca en alguna de las otras voces.

```
rule "#03 cuando un acorde no aparece en alguna de las voces se
elimina para que no pueda ser elegido"
      salience 85
      when
            $posible : NotaPosible($vozPosible : voz,
$posicionPosible : posicion, $acordesPosible : acordes)
            not(NotaPosible(voz!=$vozPosible,
posicion==$posicionPosible, acordes==$acordesPosible))
      then
            update($posible);
            retract($posible);
            System.out.println("03 se ha eliminado el valor " +
$posible.toString());
end
```

Figura 64. Regla que elimina los acordes que no aparece en las otras voces

Una vez aplicadas las reglas técnicas para llegar a una solución que no dependen directamente de las de armonía, se dispone de una serie de ellas que sí que tienen correspondencia directa con ellas. En primer lugar tenemos una regla que hace que en la medida de lo posible los acordes aparezcan completos (los tres nombres correspondientes al acorde). Para ello si hay alguna nota posible que no aparezca en el resto de las voces se pasa a final. La prioridad de esta regla no es de las más altas porque hay otras más importantes que se tienen que cumplir antes de llegar a esta solución.

```
rule "#04 Si un valor posible aparece una unica vez en el acorde esa es la
nota que hay que poner en esa voz"
      salience 45
      when
            $posible : NotaPosible($vozPosible : voz, $posicionPosible :
      posicion, $nombrePosible : nombre)
            not (NotaPosible(posicion==$posicionPosible,
      nombre==$nombrePosible))
```

```
            $posible2 : NotaPosible(voz==$vozPosible,
      posicion==$posicionPosible, nombre!=$nombrePosible)
      then
            retract($posible2);
            System.out.println("04 se ha eliminado el valor " +
      $posible2.toString());
end
```

Figura 65. Regla que fuerza a que todas las notas del acorde aparezcan en cada posición

Una regla armónica que hay que cumplir con mayor prioridad que las anteriores es que la distancia entre soprano y contralto, así como la de contralto y soprano, no superen la distancia de una octava, que numéricamente se traduce en que la diferencia de las notas en valor MIDI no supere el valor 12.

```
rule "#05 La distancia entre soprano y contralto no puede ser mayor de una
octava"
      salience 89
      when
            $resuelto : NotaFinal(voz==Nota.SOPRANO, $posicionResuelta :
      posicion)
            $posible : NotaPosible(voz==Nota.CONTRALTO,
      posicion==$posicionResuelta)
            eval((($resuelto.getOctava()-
      $posible.getOctava())*12+$resuelto.getNombre()-
      $posible.getNombre())>12);
      then
            retract ($posible);
            System.out.println("05 se ha eliminado el valor " +
      $posible.toString());
end

rule "#06 La distancia entre contralto y tenor no puede ser mayor de una
octava. Tenor posible"
      salience 89
      when
            $resuelto : NotaFinal(voz==Nota.CONTRALTO, $posicionResuelta :
      posicion)
            $posible : NotaPosible(voz==Nota.TENOR,
      posicion==$posicionResuelta)
            eval((($resuelto.getOctava()-
      $posible.getOctava())*12+$resuelto.getNombre())-
      $posible.getNombre())>12);
      then
            retract ($posible);
            System.out.println("06 se ha eliminado el valor " +
      $posible.toString());
end

rule "#07 La distancia entre contralto y tenor no puede ser mayor de una
octava. Contralto posible"
```

```
	salience 89
	when
		$resuelto : NotaFinal(voz==Nota.TENOR, $posicionResuelta :
	posicion)
		$posible : NotaPosible(voz==Nota.CONTRALTO,
	posicion==$posicionResuelta)
		eval((($posible.getOctava()-
	$resuelto.getOctava())*12+$posible.getNombre())-
	$resuelto.getNombre())>12);
	then
		retract($posible);
		System.out.println("07 se ha eliminado el valor " +
	$posible.toString());
end
```

Figura 66. Reglas para controlar la distancia permitida entre voces

En el caso de soprano y contralto, sólo hay que tener en cuenta la posibilidad de que la nota final sea el soprano y la posible la contralto, porque la voz soprano es la que se usa como entrada al sistema y no puede cambiarse, por lo que todas las notas pertenecientes a esa voz son un objeto de NotaFinal. En el caso de contralto y tenor, sí que hay que tener en cuenta que la nota final puede pertenecer a cualquiera de las dos voces y la que hay que modificar es la que aún no tiene decisión sobre la solución. Por esta razón se emplean dos reglas ya que de esta forma se tienen en cuenta las dos posibilidades.

Otra premisa que las voces deben cumplir es que la voz superior nunca puede estar por debajo de la inferior y viceversa. Para ello se tienen en cuenta los dos casos que se pueden dar, que la voz superior sea final y la inferior posible, y el contrario, que sea la inferior la final.

```
rule "#08 Las voces no se pueden cruzar. Resuelta la voz superior la
inferior no esté por encima"
	salience 85
	when
		$resuelto : NotaFinal($posicionResuelta : posicion)
		$posible : NotaPosible(posicion==$posicionResuelta)
		eval($resuelto.getVoz()-$posible.getVoz()==1);
		eval(($posible.getOctava()<$resuelto.getOctava()) ||
	(($posible.getOctava()==$resuelto.getOctava()) &&
	($posible.getNombre()>$resuelto.getNombre())));
	then
		retract ($posible);
```

```
            System.out.println("08 se ha eliminado el valor " +
      $posible.toString());
end

rule "#09 Las voces no se pueden cruzar. Resuelta voz inferior la superior
no esté por debajo"
      salience 85
      when
            $resuelto : NotaFinal($posicionResuelta : posicion)
            $posible : NotaPosible(posicion==$posicionResuelta)
            eval($resuelto.getVoz()-$posible.getVoz() == -1);
            eval(($posible.getOctava()>$resuelto.getOctava()) ||
      (($posible.getOctava()==$resuelto.getOctava()) &&
      ($posible.getNombre()<$resuelto.getNombre())));
      then
            retract ($posible);
            System.out.println("09 se ha eliminado el valor " +
      $posible.toString());
end
```

Figura 67. Reglas para que las voces no se crucen

Otra de las comprobaciones que hay que hacer es que las voces sean conjuntas ya que si esto se cumple entre todas las voces conjuntas no se dará el caso de que una voz superior se cruce con una inferior. También hay que controlar qué voz es la superior, esto se lleva a cabo mediante la resta de voces, ya que si la resuelta menos la posible da 1(las voces se identifican con números enteros en el sistema), la resuelta se encontrará inmediatamente por encima de la posible y lo contrario en el caso de -1.

Para que no haya dos distancias consecutivas entre las mismas voces hay una serie de reglas en las que se comprueba la posición siguiente y anterior para cada combinación además de tener en cuenta qué voz es la que no está resuelta. En el caso de la voz de la soprano sólo hay que tener en cuenta que sean las otras voces las que son posibles ya que la voz soprano tendrá siempre notas finales.

```
rule "#10 quintas consecutivas soprano-contralto. Nota siguiente"
      salience 50
      when
            $resuelto : NotaFinal(voz==Nota.SOPRANO, $posicionResuelta :
      posicion)
            $resuelto2 : NotaFinal(voz==Nota.CONTRALTO,
      posicion==($posicionResuelta))
            $resuelto3 : NotaFinal(voz==Nota.SOPRANO,
      posicion==($posicionResuelta + 1))
            $posible : NotaPosible(voz==Nota.CONTRALTO, posicion ==
      ($posicionResuelta + 1))
```

```
            eval(((12+$resuelto.getNombre()-$resuelto2.getNombre())==7) ||
      ((12+$resuelto.getNombre()-$resuelto2.getNombre())==19));
            eval(((12+$resuelto3.getNombre()-$posible.getNombre())==7) ||
      ((12+$resuelto3.getNombre()-$posible.getNombre())==19));
      then
            retract($posible);
            System.out.println("10 Se ha eliminado el valor " +
      $posible.toString());
end

rule "#11 quintas consecutivas soprano-contralto. Nota anterior"
      salience 50
      when
            $resuelto : NotaFinal(voz==Nota.SOPRANO, $posicionResuelta :
      posicion)
            $resuelto2 : NotaFinal(voz==Nota.CONTRALTO,
      posicion==$posicionResuelta)
            $resuelto3 : NotaFinal(voz==Nota.SOPRANO,
      posicion==($posicionResuelta-1))
            $posible : NotaPosible(voz==Nota.CONTRALTO, posicion==
      ($posicionResuelta-1))
            eval(((12+$resuelto.getNombre()-$resuelto2.getNombre()==7)) ||
      ((12+$resuelto.getNombre()-$resuelto2.getNombre())==19));
            eval(((12+$resuelto3.getNombre()-$posible.getNombre())==7) ||
      ((12+$resuelto3.getNombre()-$posible.getNombre())==19));
      then
            retract($posible);
      System.out.println("11 Se ha eliminado el valor " +
      $posible.toString());
end
```

Figura 68. Reglas quintas consecutivas entre soprano y contralto

Estas reglas existen también para las combinaciones soprano-tenor y soprano-bajo de la misma manera (reglas 12, 13, 14 y 15). Las quintas se calculan igual independientemente de la octava a la que pertenezcan ya que aquí lo que importa es el nombre. El factor de corrección 12 se aplica porque se puede dar el caso que aun la nota encontrándose por encima una de otra, el valor de su nombre se encuentre por debajo, y si no se aplicase no se detectaría la quinta. Teniendo en cuenta ese factor la distancia de quinta puede ser equivalente a 7 o a 19. En esta regla no se tiene en cuenta la posibilidad de que la voz superior se encuentre por debajo de la inferior porque la regla encargada de eso tiene una prioridad mayor que ésta.

En el caso del resto de combinaciones hay que tener en cuenta que ambas voces pueden ser posibles por lo que el numero de reglas se ve duplicado. El resto de

consideraciones son iguales. Estas relaciones de voces son contralto-tenor (reglas 16 - 19), contralto-bajo (reglas 20 - 23), tenor-bajo (reglas 24 - 27). Al tener en cuenta que cualquiera de las voces puede ser sobre la que hay que decidir se ven cubiertas todas las posibilidades.

```
rule "#16 quintas consecutivas contralto-tenor posible tenor. Nota
siguiente"
      salience 50
      when
            $resuelto : NotaFinal(voz==Nota.CONTRALTO, $posicionResuelta :
      posicion)
            $resuelto2 : NotaFinal(voz==Nota.TENOR,
      posicion==$posicionResuelta)
            $resuelto3 : NotaFinal(voz==Nota.CONTRALTO,
      posicion==($posicionResuelta+1))
            $posible : NotaPosible(voz==Nota.TENOR,
      posicion==($posicionResuelta+1))
            eval(((12+$resuelto.getNombre()-$resuelto2.getNombre())==7) ||
      ((12+$resuelto.getNombre()-$resuelto2.getNombre())== 19));
            eval(((12+$resuelto3.getNombre()-$posible.getNombre())==7) ||
      ((12+$resuelto3.getNombre()-$posible.getNombre())== 19));
      then
            retract($posible);
      System.out.println("16 Se ha eliminado el valor " +
      $posible.toString());
end

rule "#17 quintas consecutivas contralto-tenor posible tenor. Nota
anterior"
      salience 50
      when
            $resuelto : NotaFinal(voz==Nota.CONTRALTO, $posicionResuelta :
      posicion)
            $resuelto2 : NotaFinal(voz==Nota.TENOR,
      posicion==$posicionResuelta)
            $resuelto3 : NotaFinal(voz==Nota.CONTRALTO,
      posicion==($posicionResuelta - 1))
            $posible : NotaPosible(voz==Nota.TENOR,
      posicion==($posicionResuelta - 1))
            eval(((12+$resuelto.getNombre()-$resuelto2.getNombre())==7) ||
      ((12+$resuelto.getNombre()-$resuelto2.getNombre())==19));
            eval(((12+$resuelto3.getNombre()-$posible.getNombre())==7) ||
      ((12+$resuelto3.getNombre()-$posible.getNombre())==19));
      then
            retract($posible);
      System.out.println("17 Se ha eliminado el valor " +
      $posible.toString());
end

rule "#18 quintas consecutivas contralto-tenor posible contralto. Nota
siguiente"
      salience 50
      when
            $resuelto : NotaFinal(voz==Nota.CONTRALTO, $posicionResuelta :
      posicion)
```

```
            $resuelto2 : NotaFinal(voz==Nota.TENOR,
      posicion==$posicionResuelta)
            $resuelto3 : NotaFinal(voz==Nota.TENOR,
      posicion==($posicionResuelta+1))
            $posible : NotaPosible(voz==Nota.CONTRALTO,
      posicion==($posicionResuelta+1))
            eval(((12+$resuelto.getNombre()-$resuelto2.getNombre())== 7)
      || ((12+$resuelto.getNombre()-$resuelto2.getNombre())==19));
            eval(((12+$posible.getNombre()-$resuelto3.getNombre())== 7) ||
      ((12+$posible.getNombre()-$resuelto3.getNombre())==19));
      then
            retract($posible);
      System.out.println("18 Se ha eliminado el valor " +
      $posible.toString());
end

rule "#19 quintas consecutivas contralto-tenor posible contralto. Nota
anterior"
      salience 50
      when
            $resuelto : NotaFinal(voz==Nota.CONTRALTO, $posicionResuelta :
      posicion)
            $resuelto2 : NotaFinal(voz==Nota.TENOR,
      posicion==$posicionResuelta)
            $resuelto3 : NotaFinal(voz==Nota.TENOR,
      posicion==($posicionResuelta - 1))
            $posible : NotaPosible(voz==Nota.CONTRALTO,
      posicion==($posicionResuelta - 1))
            eval(((12+$resuelto.getNombre()-$resuelto2.getNombre())==7) ||
      ((12+$resuelto.getNombre()-$resuelto2.getNombre())==19));
            eval(((12+$posible.getNombre()-$resuelto3.getNombre())==7) ||
(     (12+$posible.getNombre()-$resuelto3.getNombre())==19));
      then
            retract($posible);
      System.out.println("19 Se ha eliminado el valor " +
      $posible.toString());
end
```

Figura 69. Ejemplo de regla de quinta consecutiva. Contralto-tenor

Las mismas consideraciones hay que tener en cuenta para el caso de las octavas (reglas 28 - 45) con la salvedad que no hay que aplicar ningún factor de corrección dado que en las octavas se cumple que los nombres son iguales y por tanto lo que hay que hacer es comparar el valor nombre de cada nota.

```
rule "#28 octavas consecutivas soprano-contralto posible contralto. Nota
siguiente"
      salience 50
      when
            $resuelto : NotaFinal (voz==Nota.SOPRANO, $posicionResuelta :
      posicion)
```

```
            $resuelto2 : NotaFinal (voz==Nota.CONTRALTO,
      posicion==$posicionResuelta)
            $resuelto3 : NotaFinal (voz==Nota.SOPRANO,
      posicion==($posicionResuelta+1))
            $posible : NotaPosible (voz==Nota.CONTRALTO,
      posicion==($posicionResuelta+1))
            eval($resuelto.getNombre()==$resuelto2.getNombre());
            eval($resuelto3.getNombre()==$posible.getNombre());
      then
            retract($posible);
      System.out.println("28 Se ha eliminado el valor " +
      $posible.toString());
end
```

Figura 70. Ejemplo de regla de octavas consecutivas. Soprano-contralto

Las obras empiezan por tónica, es decir, el acorde perteneciente al grado I, por lo que el acorde de armonización usado para esta posición ha de ser el indicado. Para ello se eliminan todas las notas posibles de la primera posición (que se corresponde a la posición 0 en el sistema). No se ponen más restricciones a esto ya que cualquier nota perteneciente a ese acorde puede ser válida a priori en cualquier voz. Es de máxima prioridad ya que esta posición nunca podrá ser armonizada por otro acorde.

```
rule "#46 si es el primero de la composición eliminar todas las que no
pertenezcan a I"
      salience 100
      when
            $posible : NotaPosible (posicion==0, acordes!=Nota.I)
      then
            retract($posible);
      System.out.println("46 Se ha eliminado el valor " +
      $posible.toString());
end
```

Figura 71. Regla para que el primer acorde sea de grado I

Según las normas de coral clásica, el acorde de sensible (VII) no se puede usar para la armonización, por lo que debe ser eliminada de las soluciones posibles. Este acorde se ha metido en el sistema para que sea lo más general posible ya que no se utilice en este criterio no implica que otro usuario quiera aplicar unas reglas distintas que permitan la utilización de este grado.

```
rule "#47 se elimina siempre el acorde VII"
      salience 90
```

```
      when
            $posible : NotaPosible (acordes==Nota.VII)
      then
            retract($posible);
      System.out.println("47 Se ha eliminado el valor " +
      $posible.toString());
end
```

Figura 72. Regla para la eliminación del acorde VII

Para llevar a cabo el esquema de enlaces expuesto en el apartado 3.1. hay que tener en cuenta desde dónde se puede llegar a un acorde y a dónde se puede ir desde él, además de si es el único que aparece como posible o está como definitivo (si el acorde es debido a que hay al menos una nota final o si es porque es el único acorde posible que queda ya que el tipo). Quedarán condicionados tanto el acorde anterior como el siguiente puesto que no todas las combinaciones están permitidas.

El primer acorde es un caso especial, ya que desde él se puede ir a cualquier acorde y por tanto no se ponen condiciones. Sin embargo a ese acorde sólo se puede llegar desde el grado II, IV y V. Por eso unicamente hay que tener en cuenta la posición anterior a ese acorde.

```
rule "#48 al grado I se puede llegar desde II, IV y V. Solo este acorde es
posible"
      salience 40
      when
            $posible : NotaPosible($posicionPosible : posicion,
(acordes==Nota.I))
            not ((NotaPosible(posicion==$posicionPosible,
acordes==Nota.II)))
            not (NotaPosible(posicion==$posicionPosible,
acordes==Nota.III))
            not (NotaPosible(posicion==$posicionPosible,
acordes==Nota.IV))
            not (NotaPosible(posicion==$posicionPosible, acordes==Nota.V))
            not (NotaPosible(posicion==$posicionPosible,
acordes==Nota.VI))
            not (NotaPosible(posicion==$posicionPosible,
acordes==Nota.VII))
            $posible2 : NotaPosible(posicion==($posicionPosible-1),
(acordes==Nota.I || acordes==Nota.III || acordes==Nota.VI ||
acordes==Nota.VII))
      then
            update($posible2);
            retract($posible2);
```

```
            System.out.println("48 Se ha eliminado el valor " +
$posible2.toString());
end

rule "#49 al grado I se puede llegar desde II, IV y V. Acorde resuelto"
      salience 40
      when
            $resuelto : NotaFinal($posicionFinal : posicion, acordes ==
Nota.I)
            $posible : NotaPosible(posicion==($posicionFinal-1),
(acordes== Nota.I || acordes==Nota.III || acordes==Nota.VI ||
acordes==Nota.VII))
      then
            update($posible);
            retract($posible);
            System.out.println("49 Se ha eliminado el valor " +
$posible.toString());
end
```

Figura 73. Reglas de enlaces con el grado I

En el grado II nos encontramos dos casos, estar en el grado II con lo que se condiciona la posición anterior a los grados I, III y V, y llegar al grado II que condiciona los enlaces anteriores a II y IV. En cada caso hay que tener en cuenta que el acorde se puede dar porque sólo quede ese acorde como posible o porque haya al menos una nota final.

```
rule "#50 cuando se está en el grado II se puede pasar a I, III y V. Solo
ese acorde posible"
      salience 40
      when
            $posible : NotaPosible($posicionPosible : posicion,
(acordes==Nota.II))
            not ((NotaPosible(posicion==$posicionPosible,
acordes==Nota.I)))
            not (NotaPosible(posicion==$posicionPosible,
acordes==Nota.III))
            not (NotaPosible(posicion==$posicionPosible,
acordes==Nota.IV))
            not (NotaPosible(posicion==$posicionPosible, acordes==Nota.V))
            not (NotaPosible(posicion==$posicionPosible,
acordes==Nota.VI))
            not (NotaPosible(posicion==$posicionPosible,
acordes==Nota.VII))
            $posible2 : NotaPosible(posicion==($posicionPosible+1),
(acordes==Nota.II || acordes==Nota.VI || acordes==Nota.VI ||
acordes==Nota.VII))
      then
            update($posible2);
            retract($posible2);
```

```
            System.out.println("50 Se ha eliminado el valor " +
$posible2.toString());
end

rule "#51 cuando se está en el grado II se puede pasar a I, III y V.
Acorde resuelto"
      salience 40
      when
            $resuelto : NotaFinal($posicionFinal : posicion, acordes ==
Nota.II)
            $posible : NotaPosible(posicion==($posicionFinal+1),
(acordes== Nota.II || acordes==Nota.IV || acordes==Nota.VI ||
acordes==Nota.VII))
      then
            update($posible);
            retract($posible);
            System.out.println("52 Se ha eliminado el valor " +
$posible.toString());
end

rule "#53 al grado II se puede llegar desde III y VI. Solo ese acorde
posible"
      salience 40
      when
            $posible : NotaPosible($posicionPosible : posicion,
(acordes==Nota.II))
            not ((NotaPosible(posicion==$posicionPosible,
acordes==Nota.I)))
            not (NotaPosible(posicion==$posicionPosible,
acordes==Nota.III))
            not (NotaPosible(posicion==$posicionPosible,
acordes==Nota.IV))
            not (NotaPosible(posicion==$posicionPosible, acordes==Nota.V))
            not (NotaPosible(posicion==$posicionPosible,
acordes==Nota.VI))
            not (NotaPosible(posicion==$posicionPosible,
acordes==Nota.VII))
            $posible2 : NotaPosible(posicion==($posicionPosible-1),
(acordes==Nota.I || acordes==Nota.II || acordes==Nota.VI ||
acordes==Nota.V || acordes==Nota.VII))
      then
            update($posible2);
            retract($posible2);
            System.out.println("53 Se ha eliminado el valor " +
$posible2.toString());
end

rule "#54 al grado II se puede llegar desde III y VI. Acorde resuelto"
      salience 40
      when
            $resuelto : NotaFinal($posicionFinal : posicion, acordes ==
Nota.II)
            $posible : NotaPosible(posicion==($posicionFinal-1),
(acordes==Nota.I || acordes== Nota.II || acordes==Nota.VI ||
acordes==Nota.V || acordes==Nota.VII))
      then
            update($posible);
```

```
            retract($posible);
            System.out.println("54 Se ha eliminado el valor " +
$posible.toString());
end
```

Figura 74. Reglas de enlaces condicionados por el grado II

El resto de enlaces se hacen de la misma forma que los del grado II, siguiendo la relación mostrada en la figura 32. Estas reglas comprenden desde la número 50 a la número 70 teniendo todas una prioridad 40.

Otra regla necesaria para llegar a un resultado satisfactorio es que cuando se decida que un acorde pertenece a una posición, es decir, cuando se pasa a final una nota que a su vez lleva asociado un acorde, se eliminan el resto de acordes de las posibles para esa posición. Esta regla debe tener mayor prioridad que los enlaces puesto que en caso contrario se podría eliminar alguna nota de este acorde.

```
rule "#71 si se elige un acorde eliminar el resto"
      salience 90
      when
            $resuelto : NotaFinal($vozPosible : voz, $posicionPosible :
      posicion, $acordesPosible : acordes)
            $posible : NotaPosible(voz!=$vozPosible, posicion==posicion,
      acordes!=$acordesPosible)
      then
            retract($posible);
      System.out.println("71 Se ha eliminado el valor " +
      $posible.toString());
end
```

Figura 75. Regla que elimina el resto de acordes cuando se elige uno

Por último se contempla el caso en el que no se llegue a una nota final con las reglas anteriores, lo que quiere decir que cualquiera de las soluciones posibles que quedan es válida. Por ello se tiene una regla que pasa una de las notas a final y avisa a la memoria de trabajo para que ejecute de nuevo las reglas con ese nuevo resultado que se ha elegido ya que no todas las combinaciones cumplirían todas las reglas. La prioridad es la más baja del sistema porque solo se debe activar cuando el sistema se estanca sin una solución final.

```
rule "#72 cuando hay que decidir aleatoriamente"
      salience 10
            when
                  $posible : NotaPosible()
            then
                  update($posible);
                  retract($posible);
                  insert(new NotaFinal($posible));
            System.out.println("72 Se ha resuelto la nota " +
      $posible.toString());
end
```

Figura 76. Regla de elección aleatoria

4.2. Aleatorio

Las reglas de este archivo se han seleccionado del anterior de forma que el resultado sea aleatorio. Para ello se han utilizado las reglas que acotan las tesituras de las voces, la regla que fuerza a que en caso de que una nota se pase a final el resto de las posibles sean eliminadas, y la que elige pasar a final notas aleatorias. De esta forma se deja que sea el sistema el que, sin más condicionantes, elija la salida que se dé por resultado.

```
rule "Tesitura voz contralto"
      salience 100
      when
            $posible : NotaPosible(voz==Nota.CONTRALTO, octava>4 ||
      (octava==4 && nombre>Nota.B) || (octava==3 && nombre<Nota.B ||
      octava<3))
      then
            update($posible);
            retract($posible);
            System.out.println("TesCont Se ha eliminado la nota posible "
      + $posible.toString());
end

rule "Tesitura voz tenor"
      salience 100
      when
            $posible : NotaPosible(voz==Nota.TENOR, (octava>4) ||
      (octava==4 && nombre>Nota.F) || (octava<3) || (octava==3 &&
      nombre<Nota.E))
      then
            retract($posible);
            System.out.println("TesTen Se ha eliminado la nota posible " +
      $posible.toString());
```

```
end

rule "Tesitura voz bajo"
	salience 100
	when
		$posible : NotaPosible(voz==Nota.BAJO, octava>3 || (octava==3
	&& nombre>Nota.A) || octava<2 || (octava==2 && nombre<Nota.G))
	then
		retract($posible);
		System.out.println("TesBaj Se ha eliminado la nota posible " +
	$posible.toString());
end

rule "#01 Una vez se resuelve una nota se eliminan los valores posibles de
esa posicion"
	salience 90
	when
		$resuelto : NotaFinal($vozResuelta : voz, $posicionResuelta :
	posicion)
		$posible : NotaPosible(voz==$vozResuelta,
	posicion==$posicionResuelta)
	then
		retract($posible);
		update($resuelto);
		System.out.println("01 Se ha eliminado la nota" +
	$posible.toString());
end

rule "#72 cuando hay que decidir aleatoriamente"
	salience 10
		when
			$posible : NotaPosible()
		then
			update($posible);
			retract($posible);
			insert(new NotaFinal($posible));
		System.out.println("72 Se ha resuelto la nota " +
	$posible.toString());
end
```

Figura 77. Reglas del archivo aleatorio.drl

5. EVALUACIÓN

Para la evaluación de sistema se han usado cuatro archivos MIDI en distintos compases. A cada archivo se le han ejecutado los distintos estilos para comprobar que las salidas de las ejecuciones son distintas y correctas. Se hará una comparación de una salida de cada estilo para comprobar que ésta depende de las reglas aplicadas en la resolución.

5.1. Entrada Do Mayor compás 6/8

Esta obra consta de 15 compases de 6/8 en el que no todas las figuras deben ser armonizadas ya que hay notas de menor valor que el acordado como figura a armonizar, que para el caso del 6/8 es negra con puntillo. En este caso se debe armonizar solo la primera de los grupos de 3 corcheas que aparecen.

```
CM.mid CMresultado.mid CoralClasica.drl 0 M 6 8
```

Figura 78. Información de entrada del sistema CM

Como primer parámetro (CM.mid) se pone el archivo donde se encuentra la melodía a armonizar. El segundo parámetro indica en nombre del archivo en el que se quiere guardar el resultado de la armonización. A continuación se indican el archivo de reglas y la tonalidad de la melodía. La tonalidad está separada en dos parámetros, el primero es el nombre, que se corresponde con un número entero que se corresponde con los valores de las notas en la clase Nota, y el segundo la modalidad (mayor o menor), en este caso 0 M (Do mayor). Por último se indica el compás en el que está la melodía, primero el numerador y luego el denominador, en esta entrada 6/8.

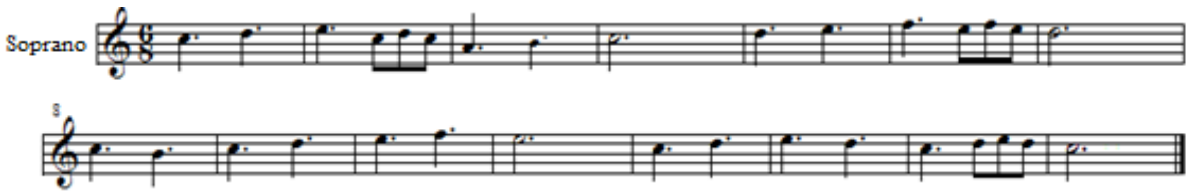

Figura 79. Partitura en Do mayor introducida al sistema

Para este caso el sistema calcula que tiene que armonizar 26 notas, que coinciden con el número de notas que cumplen el criterio de armonización, ya que de las 32 notas hay cuatro que no tienen armonización propia por tener una duración y una posición a la que no le corresponde ser metida en el sistema como nota a armonizar.

En este caso las notas se han armonizado con los enlaces de la figura 32. Esta armonía ha resultado de la activación de reglas del archivo introducido. Las reglas que se han activado son las de tesitura, la regla 46 que fuerza a que el primer acorde sea de grado I, la 47 que elimina el acorde de VII del sistema, la 9 que impide que las voces se crucen, la regla 71 que hace que todas las notas de la posición sean del mismo acorde, la 1 que elimina las notas posibles cuando se pasa una nota a definitiva, la 2 que pasa a definitiva la última nota que queda como posible, las reglas de octavas y quintas, y las reglas de enlaces.

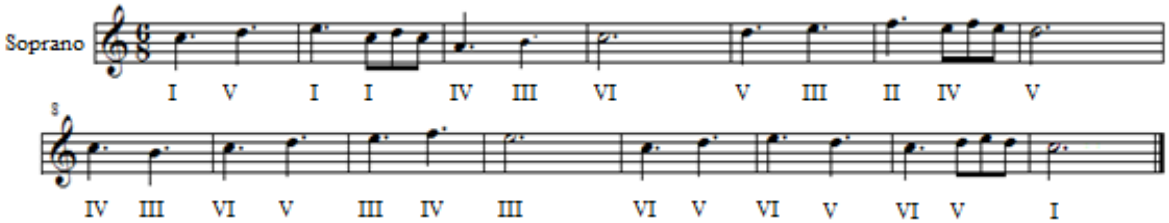

Figura 80. Análisis resultante del sistema. Do mayor

Salida del sistema:

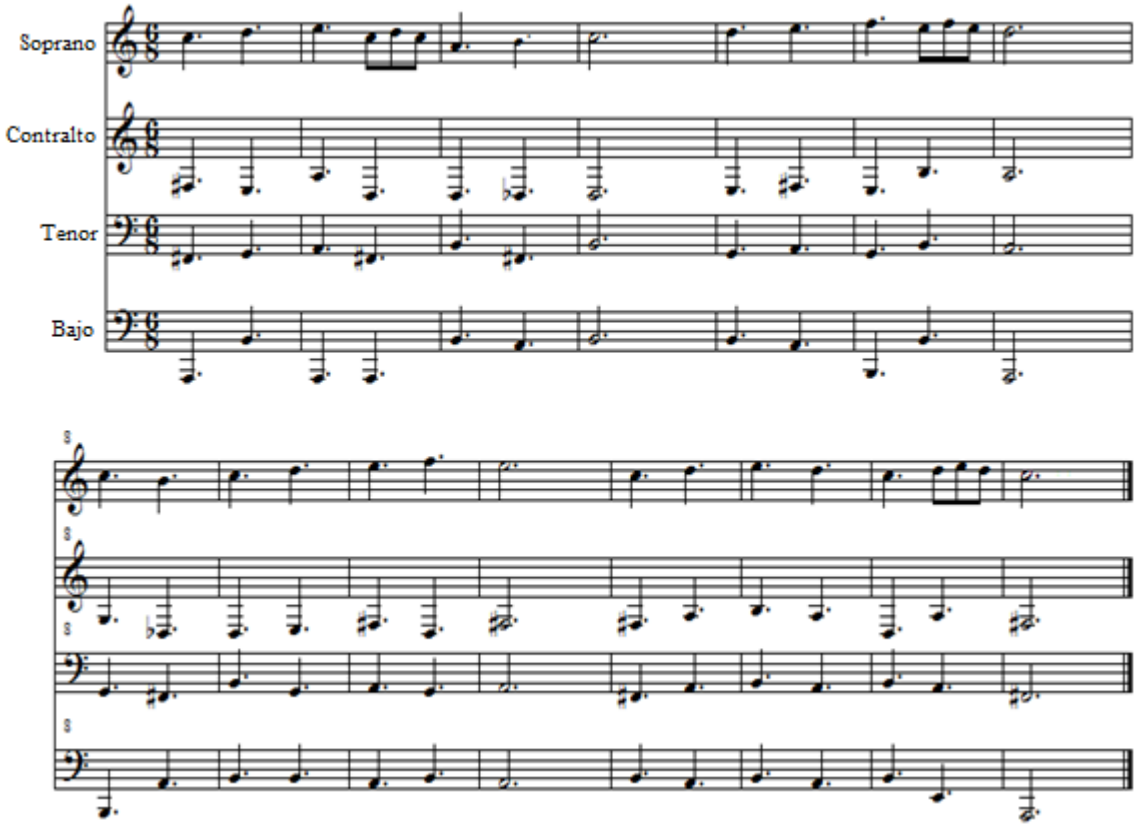

Figura 81. Salida del sistema Do mayor reglas coral clásica

Ahora se hace la misma prueba pero pasándole al sistema el archivo de reglas aleatorio.dlr. En este caso no habrá un análisis armónico resultante puesto que no se condiciona que la salida pertenezca cada posición a un acorde. La salida es distinta a la resultante de las reglas de coral clásica.

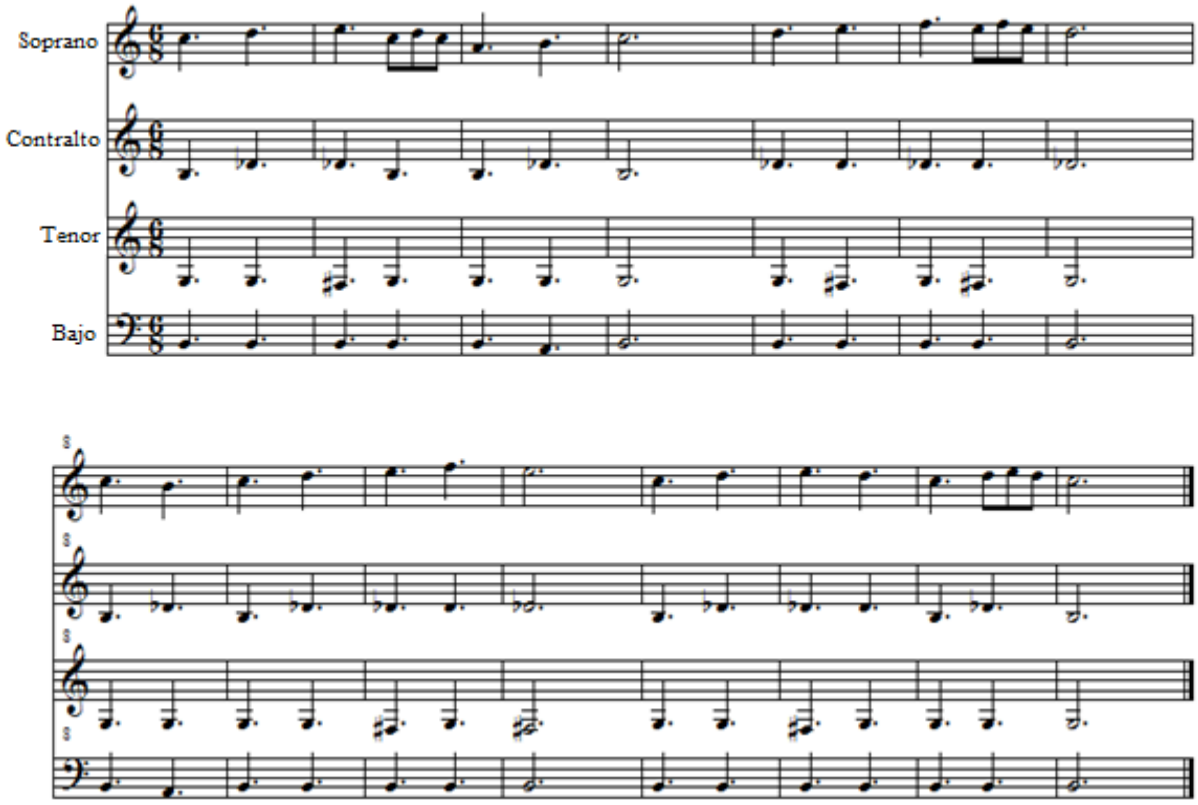

Figura 82. Salida del sistema Do mayor reglas aleatorias

5.2. Entrada Sib Mayor compás 3/4

Esta obra consta de 8 compases de 3/4 en el que en este caso tampoco todas las figuras deben ser armonizadas puesto que hay algunas que no cumplen el criterio adoptado, por el que se armoniza en múltiplos de negra.

```
BbM.mid BbMresultado.mid CoralClasica.drl 10 M 3 4
```

Figura 83. Información de entrada al sistema BbM

El primer parámetro (BbM.mid) indica el archivo MIDI que se va a armonizar. El segundo (BbMresultado.mid) es en el que se va a guardar el resultado obtenido de la armonización. A continuación se introduce el nombre del archivo de reglas usado para el

estilo de la armonización (CoralClásica.drl en el primer resultado, en el segundo aleatorio.drl). La tonalidad es Si bemol mayor, por lo que como parámetros se introducen un 10 correspondiente a Si bemol, y M correspondiente al modo mayor.

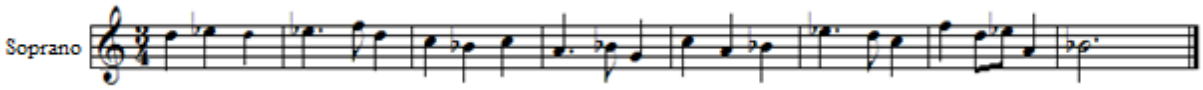

Figura 84. Partitura en Sib mayor introducida al sistema

Para este caso el sistema calcula que tiene que armonizar 19 notas, que coinciden con el número de notas que cumplen el criterio. En este caso hay figuras (corchea con puntillo) que duran una parte y media, por lo que la armonización se llevará a cabo en la negra pero no en la corchea.

Las notas se han armonizado con la armonía de la figura 89, resultado de la activación de reglas. En este caso las reglas que se han activado son las de tesitura, la regla 1 que elimina las notas posibles cuando se pasa una nota a definitiva, la 2 que pasa a definitiva la última nota que queda como posible, la 3 que elimina las notas cuyo acorde no aparece en alguna de las voces, la 7 que no deja que la distancia entre el contralto y tenor sea mayor de una octava, las reglas 8 y 9 que no dejan que las voces se cruce, las reglas de octavas y quintas, y las reglas de enlaces, la regla 71 que elimina el resto de acordes de las otras voces cuando uno es elegido y la 72 que es la aleatoria.

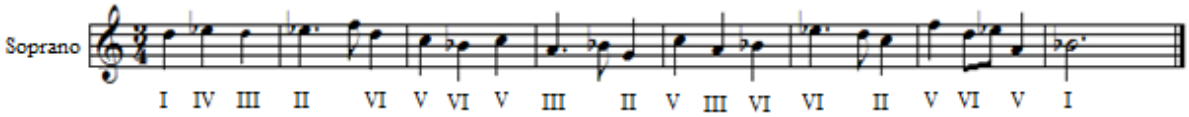

Figura 85. Análisis resultante del sistema. Sib mayor

Salida del sistema con el archivo de reglas de coral clásica:

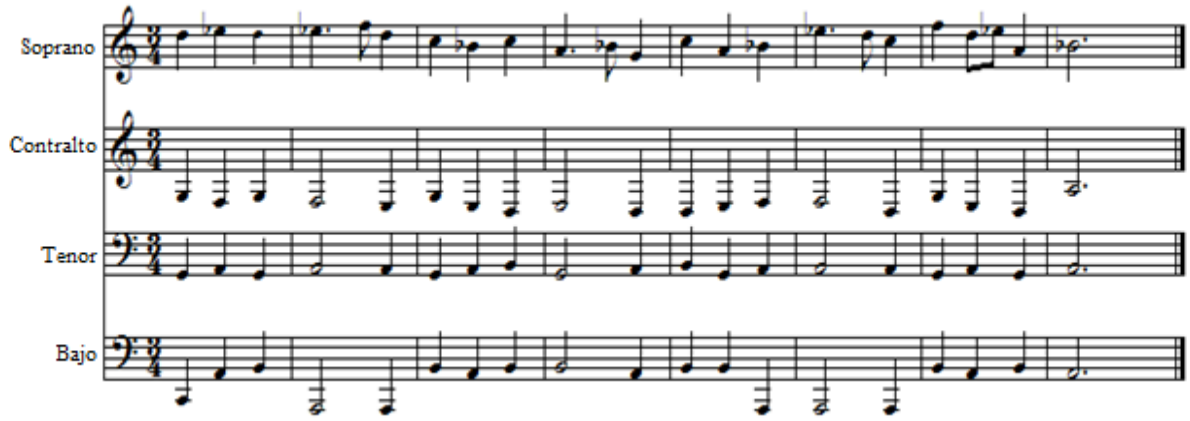

Figura 86. Salida del sistema Sib mayor reglas coral clásica

A continuación se pasa la misma entrada por el archivo de reglas aleatorias para comprobar que la salida es distinta a la anterior y no se obtiene como resultado ningún análisis, puesto que no se ha puesto como restricción que las notas pertenezcan al mismo acorde.

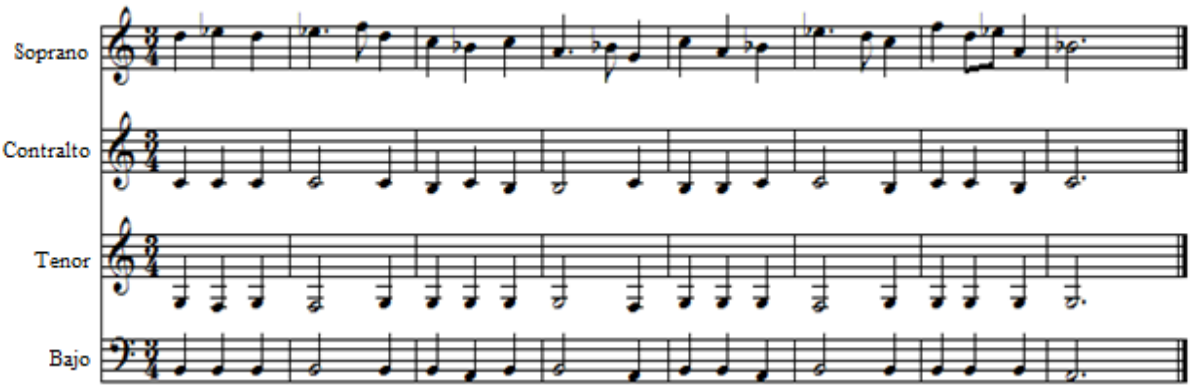

Figura 87. Salida del sistema Sib mayor reglas aleatorias

Observando ambas salidas del sistema para la misma entrada. Pero cambiando el archivo de reglas, se comprueba que la armonización obtenida es distinta acorde con dichas condiciones.

5.3. Entrada Himno de la Alegría

Este archivo es el tema principal de la obra conocida popularmente como *Himno de la Alegría*. Consta de 53 notas que cumplen el convenio de armonización en 16 compases de 4/4 (armonizar en múltiplos de negra).

```
HimnoAlegria2.mid HimnoAlegriaCCresultado.mid CoralClasica.drl 0 M 4 4
```

Figura 88. Información Himno de la Alegría introducida al sistema

El primer parámetro introducido es el nombre de la melodía que se quiere armonizar. En este caso se utiliza una melodía conocida por el nombre del *Himno de la Alegría*. A continuación se indican el archivo donde se quiere guardar el resultado y el archivo de reglas que se quiere usar para la armonización. La tonalidad de esta obra es Do mayor, con lo que los siguientes parámetros son 0 y M, y está escrita en 4/4.

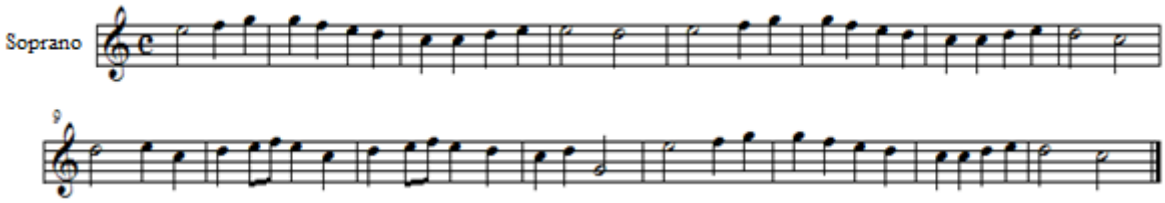

Figura 89. Partitura Himno de la alegría introducida en el sistema

Según el criterio de armonización (armonziación de las figuras que empiezan en múltiplos de negra), en esta melodía hay que armonizar 53 notas, que como se observa en la figura del análisis es el número de acordes resultantes. Las blancas al ser mayores que una negra se armonizan una sola vez y las corcheas, como son menores, se armoniza sólo la que coincide con el múltiplo de negra.

El análisis obtenido del sistema con las reglas de coral clásica es el mostrado en la siguiente figura.

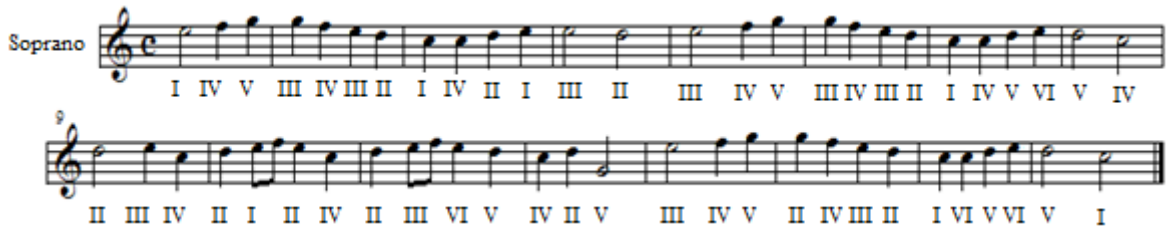

Figura 90. Análisis resultante del sistema Himno de la alegría

Al introducir en el sistema el *Himno de la alegría* y el archivo de reglas correspondiente al estilo de coral clásica se activan las reglas de tesitura, la 1 que elimina los valores posibles cuando se toma una decisión, la 2 que pasa a final una nota cuando es la única posible que queda, la 3 que elimina las notas de un acorde cuando no aparece en alguna de las voces, 6 y 7 que hacen que la distancia entre contralto y tenor no sea mayor de una octava, 8 y 9 que son para que las voces no se crucen, reglas correspondientes a quintas y octavas consecutivas, reglas para el enlazado de acordes, la 46 que obliga a que la obra empiece en grado I, la 47 que elimina el grado VII como solución posible, la 71 que elimina el resto de acordes posibles cuando se elige uno y la 72 que es la regla aleatoria.

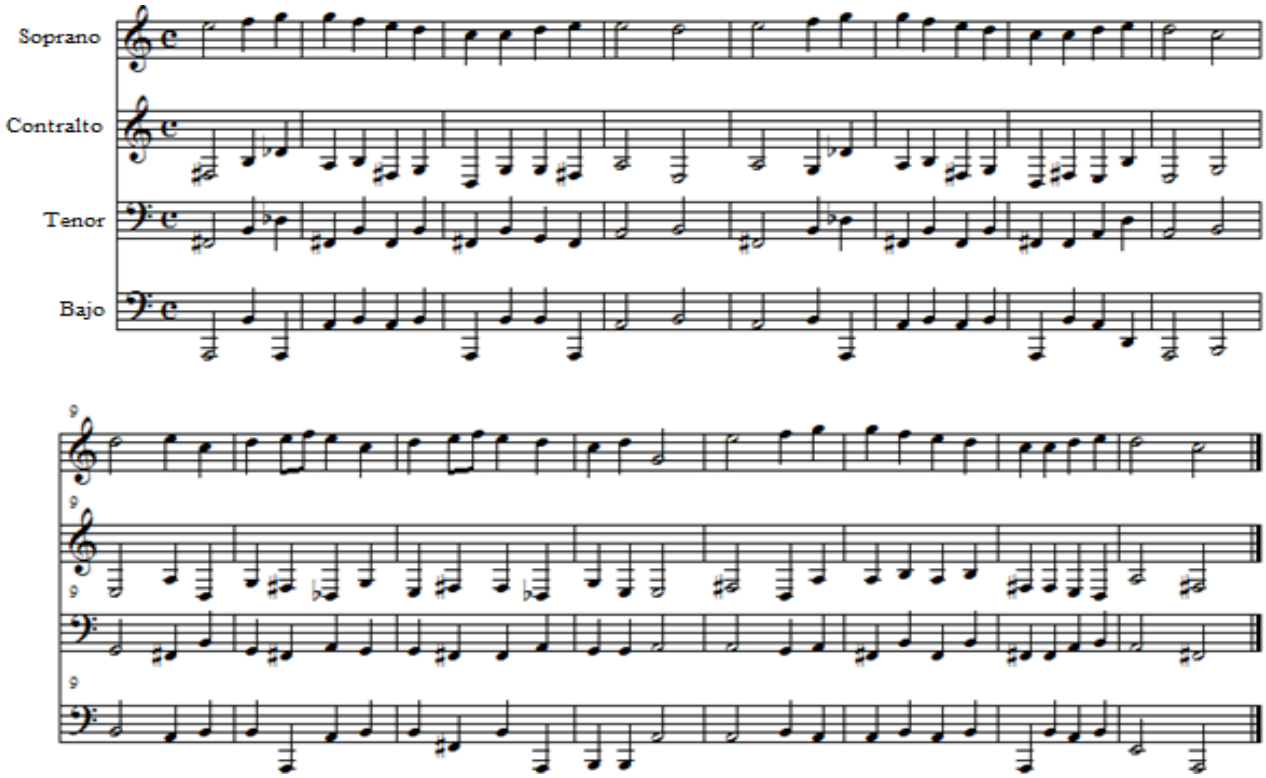

Figura 91. Salida del sistema Himno de la alegría reglas coral clásica

A continuación se hace el mismo proceso con el archivo de reglas aleatorias del que no se obtiene ningún análisis armónico debido a que no hay restricciones al respecto, aunque sí se mantiene el criterio de qué notas se armonizan y cuáles no.

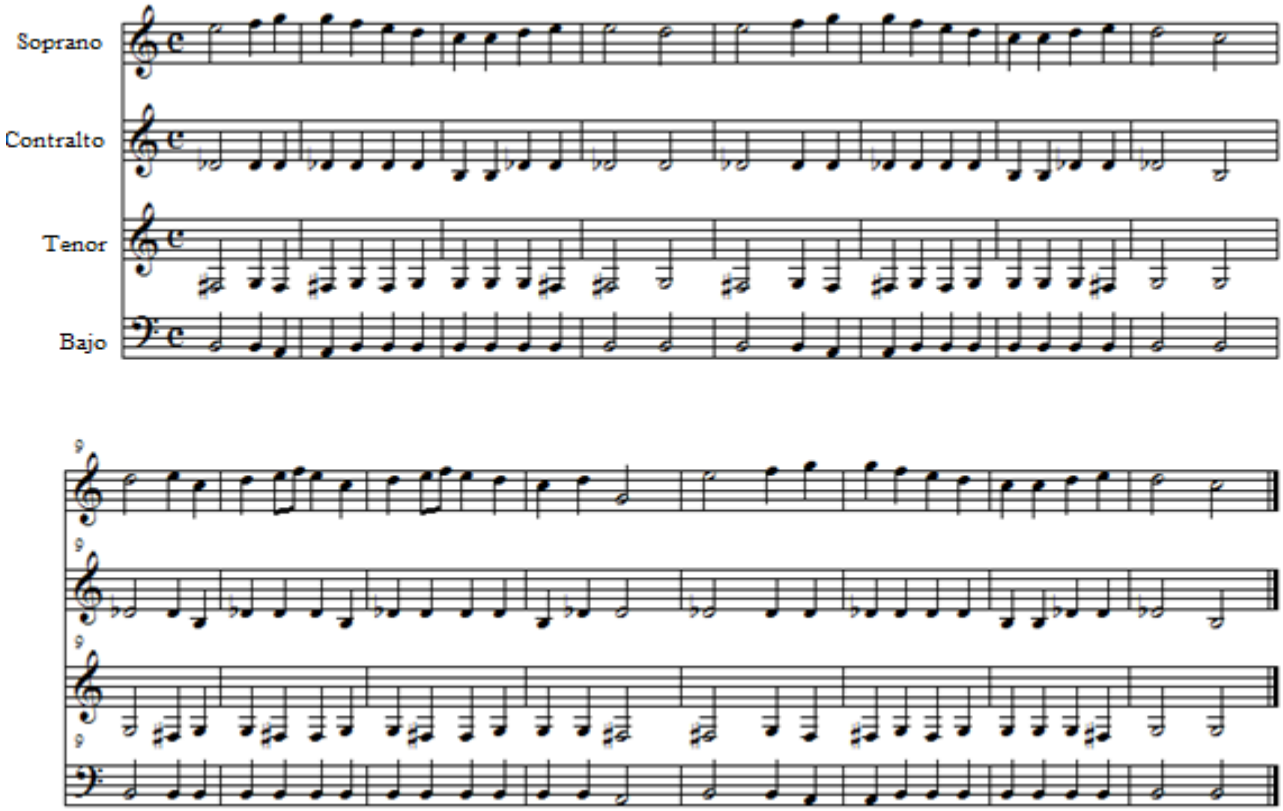

Figura 92. Salida del sistema Himno de la alegría reglas aleatorias

5.4. Entrada Bacarola

Este archivo es un arreglo de la canción conocida como *Bacarola*. Consta de 37 notas en un compás de 3/4 distribuido en 17 compases. El criterio de armonización es en múltiplos de negra.

```
bacarola.mid bacarolaCCresultado.mid CoralClasica.drl 0 M 3 4
```

Figura 93. Información Bacarola introducida al sistema

En este caso el nombre del archivo introducido al sistema es bacarola.mid, y el archivo donde se guardará bacarolaCCresultado.mid. En la primera ejecución el parámetro del archivo de reglas corresponderá al estilo de coral clásica y el segundo al aleatorio. La melodía a armonizar está en Do mayor, por lo que los parámetros siguientes serán 0 M y está en 3/4 por lo que el numerador será 3 y el denominador 4.

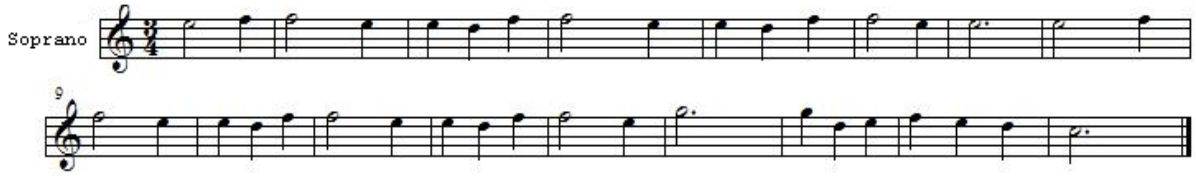

Figura 94. Partitura Bacarola introducida en el sistema

El análisis obtenido del sistema con las reglas de coral clásica es el mostrado en la siguiente figura.

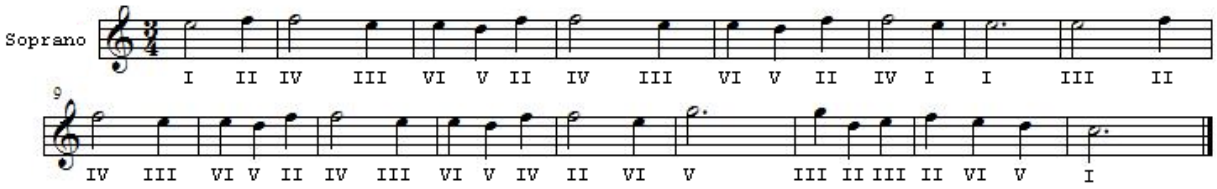

Figura 95. Análisis resultante del sistema Bacarola

Al introducir la *Bacarola* en el sistema se activan las reglas correspondientes a las tesituras, la 1 que elimina los valores posibles cuando se pasa un valor a final, la 2 que pasa una nota a final cuando es la única posible que queda, la 3 que elimina las notas que de un acorde cuando no aparece en alguna de las voces para que no queden voces vacías, 6 y 7 que hacen que la distancia entre contralto y tenor no sea mayor que una octava, 8 y 9 para que las voces no se crucen, reglas correspondientes a quintas y octavas consecutivas, reglas para el enlazado de acordes, la regla 46 que fuerza a que la obra empiece en grado I, la regla 47 que elimina el grado VII como posible solución, la regla 71 que elimina el resto de acordes como posibles cuando se elige uno y la 72 que es la regla aleatoria.

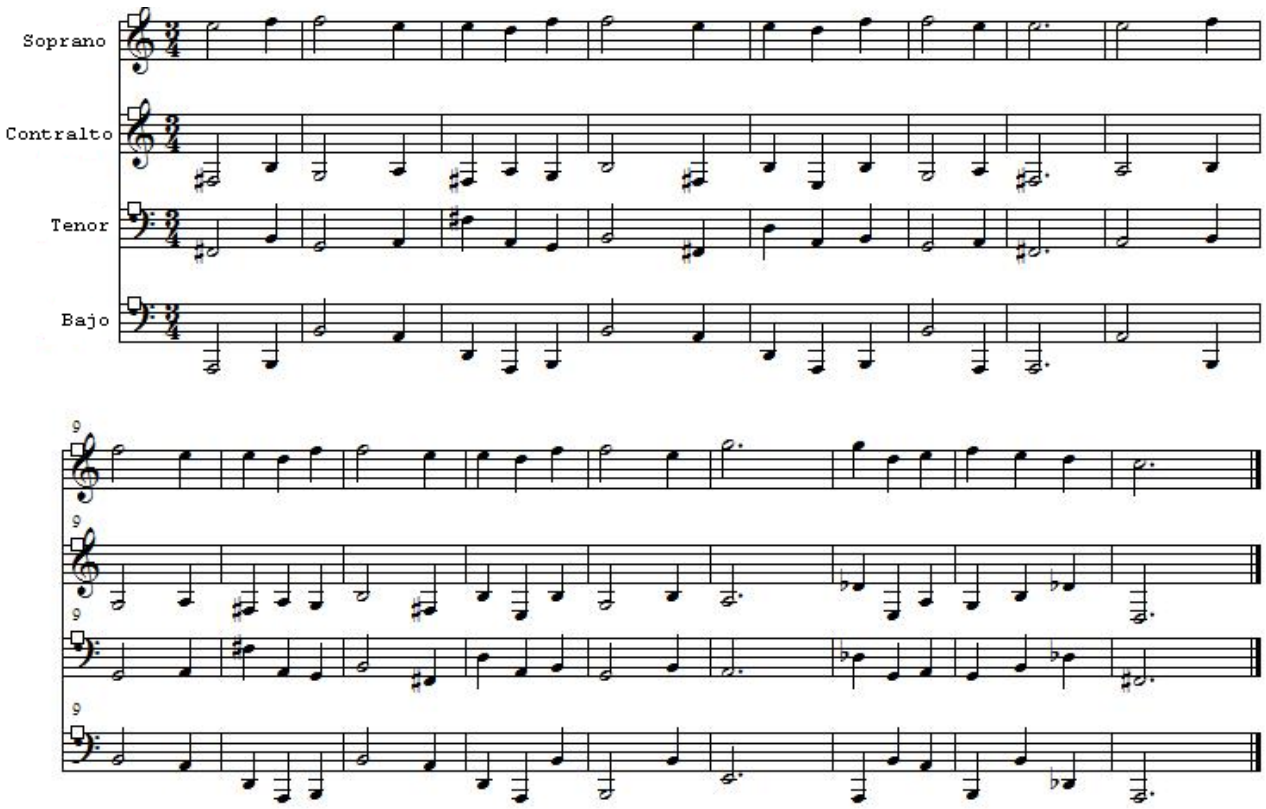

Figura 96. Salida del sistema Bacarola reglas coral clásica

A continuación se hace el mismo proceso con el archivo de reglas aleatorias del que no se obtiene ningún análisis armónico.

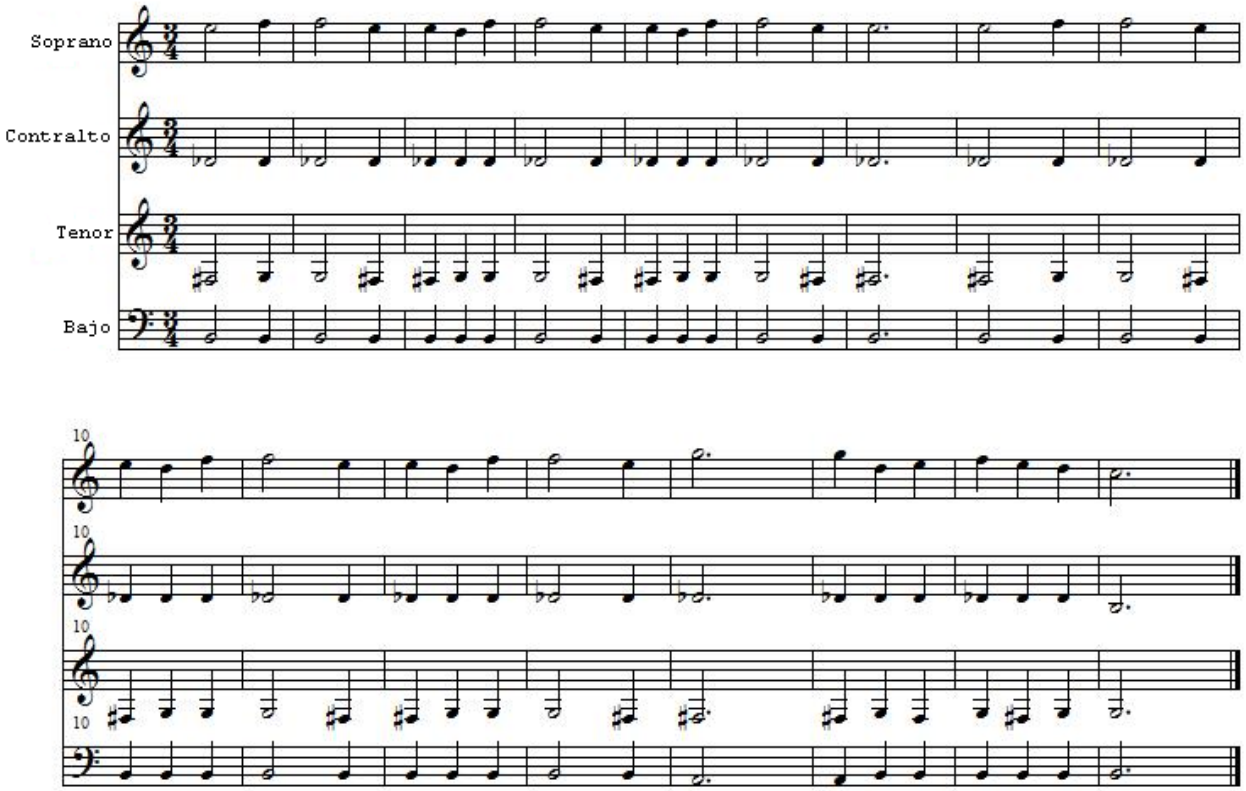

Figura 97. Salida del sistema Bacarola reglas aleatorias

6. CONCLUSIONES Y TRABAJOS FUTUROS

6.1. Conclusiones

El objetivo de este proyecto es diseñar e implementar un sistema experto para composición musical armónica en diferentes estilos a partir de conjuntos de reglas adecuados a ellos. Para lograr esto, el sistema ha de ser modular de forma que la información a armonizar y el conocimiento del estilo estén totalmente separados para que puedan ser sustituidos o modificados fácilmente.

Por esto se procesa de forma separada la melodía y el estilo, haciendo así independientes cada uno de estos módulos. La principal dificultad que radica en este punto es crear un sistema de comunicación entre ambos, ya que en las reglas se necesita información no contenida en los archivos donde se almacena música. Se ha solucionado haciendo un análisis que añada la información necesaria para el módulo de conocimiento.

Una de las decisiones más importantes en la comunicación entre los datos y el conocimiento es qué datos van a ser introducidos para la toma de decisiones. En este caso se ha optado por una forma genérica en la que se generan todas las posibilidades de forma iterativa sin poner restricciones, ya que de esta forma se consigue la separación buscada de los dos ámbitos, pues es el sistema de reglas el que impone dichas restricciones.

La información adicional necesaria para que el sistema de reglas pueda actuar son las notas que deben ser armonizadas y los acordes que se pueden usar para ello. En la parte de información, se analiza qué notas comienzan con el pulso del compás (es el criterio elegido) ya que son las que deben ser armonizadas, y para ellas se generan todas las posibles notas de armonización en las que se incluye además de la información original de la nota, el número de la nota que pueden armonizar y el grado al que pertenece.

Una vez que se tiene toda la información lista, se envía al módulo encargado del procesado de reglas. Aquí se descartan posibilidades hasta que finalmente se obtiene una solución dependiente del estilo utilizado para la armonización. La dificultad de esto se encuentra en el diseño de las reglas para crear el estilo, ya que los fallos en ellas pueden traducirse en que no haya una solución que pueda ser interpretada por el módulo de datos.

Una vez que se tiene una solución, ésta pasa al módulo de datos puesto que ya no sigue siendo necesaria la ejecución del sistema de reglas. El módulo de datos ordena y reinterpreta los datos obtenidos de las decisiones para volver a generar un archivo de audio en formato MIDI que es lo que el usuario recibe como respuesta a la entrada original. La ordenación es necesaria porque del sistema de reglas se obtienen las decisiones tomadas en el orden que han ocurrido. Se crea un nuevo archivo y no se modifica el original para que sea reutilizable con diferentes estilos sin hacer ninguna modificación.

Tras las pruebas realizadas se han sacado tres conclusiones principales. En primer lugar las melodías a armonizar deben estar siempre en la tesitura de la voz soprano sin salirse de ella. Esto no es fácil ya que la mayoría de canciones se mueven dentro de la tesitura de contralto o entre contralto y soprano. Otra conclusión es la restringida adaptabilidad de las normas de coral clásica a melodías que no pertenezcan a este estilo, ya que estas reglas están concebidas para melodías que se mueven por grados conjuntos (saltos de segunda) o con pequeños saltos (de tercera). Esto no se da en las melodías que no han sido escritas para este estilo porque limita mucho las variaciones que se pueden hacer como ocurre en el caso de las melodías del *Himno de la alegría* y *Bacarola*. Donde, al no ser obras escritas para ese estilo hay puntos donde sólo se pueden cumplir las normas más prioritarias ya que las otras no llegan a ejecutarse en dichos puntos.

Por último, debido al diseño del sistema de eliminación de posibilidades no se pueden cumplir en todo momento las reglas de menor prioridad. Habitualmente, cuando el

alumno realiza un ejercicio de armonización en el ámbito educativo llega a puntos donde no puede cumplir con las reglas, por lo que tiene que deshacer todo el trabajo realizado (o al menos parte de él) y empezar de nuevo tomando otras decisiones. En el caso de este sistema no hay opción de deshacer las decisiones tomadas por lo que se opta por seguir adelante cumpliendo las reglas que sean posibles.

6.2. Trabajos futuros

Siguiendo el mismo sistema llevado a cabo se podrían crear nuevos estilos musicales que aplicar al sistema, simplemente escribiendo nuevas bases de reglas. Estos estilos pueden ser resultado de un análisis de compositores consagrados o bien una creación del compositor siempre que cumpla que todas las posiciones tengan solución y sea única.

Una mejora que sería muy interesante es rediseñar el sistema con backtracking para poder deshacer las decisiones tomadas cuando se llegue a un punto donde alguna regla no se puede cumplir para tomar decisiones previas distintas que hagan que no se produzca de nuevo esa situación en ese punto.

Otra posible mejora al sistema presentado es la creación de un algoritmo que controle la regla aleatoria de forma que ésta dependa de un parámetro dado por el usuario. De esta forma se podría crear un estilo más personalizado.

También se puede hacer una adaptación a acordes de cuatríadas. En este caso habría que modificar el módulo de datos para que el análisis de la matriz de datos a introducir sea sobre acordes de cuatro notas y no solo de tres como está contemplado en el sistema.

El análisis armónico generalmente no se va decidiendo según se van tomando decisiones como se hace en el sistema, sino que en el ámbito educativo se lleva a cabo un análisis que tiene en cuenta las posiciones que ocupa la nota en la obra, ya que generalmente las obras suelen tener periodos de reposo llamados cadencias. Por ello se puede crear un módulo que haga un análisis previo con las posiciones relativas de las notas y a la hora de armonizar ya se tengan los enlaces decididos.

Los sistemas expertos en el ámbito de la música tienen un gran potencial. En la misma línea del sistema presentado, se podría realizar otro de finalidad didáctica para la corrección de corales realizadas en el ámbito educativo, con algunas modificaciones en el sistema de datos ya que no se trataría de crear una armonización si no de validar una ya existente, lo que ayudaría al alumno a mejorar en este área sin necesidad de depender continuamente de un profesor de apoyo para la corrección.

Fuera del presente sistema, pero como complemento a él, se pueden agregar distintos bloques que hagan que todo en conjunto sea una potente herramienta didáctica para ser usada en el ámbito del estudio profesional de la música. Estos bloques serían:

- Un sistema para las materias previas a los estudios de armonía como es el estudio de los dictados rítmicos y melódicos mediante un sistema experto de generación aleatoria de dictados.
- Un sistema de generación automática de melodía que posteriormente se podría armonizar con el sistema ya propuesto, o ser armonizado por el alumno y ser validado por un sistema de validación de coral clásica.
- Un sistema de análisis de obras. En el ámbito educativo hay materias encargadas del análisis de obras ya hechas para extraer las características de los compositores. Unido a la Inteligencia Artificial, se puede crear un sistema que aprenda del análisis de obras para autogenerar archivos de reglas, lo que permitiría a partir de un modelo generar obras del mismo estilo.

- Interfaz gráfica que facilite el uso de todos los módulos por parte del usuario.

Apéndice A: Presupuesto

UNIVERSIDAD CARLOS III DE MADRID
Escuela Politécnica Superior

PRESUPUESTO DE PROYECTO

1.- Autor:
Celia Clemente Castillo
2.- Departamento:
Ing. Telemática
3.- Descripción del Proyecto:

- Titulo: Sistema inteligente para composición armónica musical
- Duración (meses): 14

Tasa de costes Indirectos: **20%**

4.- Presupuesto total del Proyecto (valores en Euros):

25.882,6 Euros

5.- Desglose presupuestario (costes directos)

PERSONAL

Apellidos y nombre	N.I.F.	Categoría	Dedicación (personas mes) [a)]	Coste persona mes	Coste (Euro)	Firma de conformidad
Julio Villena Román		Ingeniero Senior	0,5	4.289,54	2.144,77	
Celia Clemente Castillo		Ingeniero	7	2.694,39	18.860,73	
		Personas mes 7.5		**Total**	21.005,50	

[a)] 1 Persona mes = 131,25 horas. Máximo anual de dedicación de 12 hombres mes (1575 horas)
Máximo anual para PDI de la Universidad Carlos III de Madrid de 8,8 hombres mes (1.155 horas)

EQUIPOS

Descripción	Coste (Euro)	% Uso dedicado proyecto	Dedicación (meses)	Periodo de depreciación	Coste imputable [d)]
Equipo informatico	700,00	100	14	60	163,33
				Total	163,33

[d)] Fórmula de cálculo de la Amortización:

$$\frac{\mathbf{A}}{\mathbf{B}}\mathbf{xCxD}$$

A = nº de meses desde la fecha de facturación en que el equipo es utilizado
B = periodo de depreciación (60 meses)
C = coste del equipo (sin IVA)
D = % del uso que se dedica al proyecto (habitualmente 100%)

SUBCONTRATACIÓN DE TAREAS

Descripción	Empresa	Coste imputable
Músico (40h)	Autónomo	400,00
	Total	400,00

OTROS COSTES DIRECTOS DEL PROYECTO[e)]

Descripción	Empresa	Costes imputable
Licencia no comercial de Drools		0,00
	Total	0,00

[e)] Este capítulo de gastos incluye todos los gastos no contemplados en los conceptos anteriores, por ejemplo: fungible, viajes y dietas, otros,…

6.- Resumen de costes

Presupuesto Costes Totales	**Presupuesto Costes Totales**
Personal	21005,50
Amortización	163,33
Subcontratación de tareas	400
Costes de funcionamiento	0
Costes Indirectos	4.313,77
Total	25.882,6

BIBLIOGRAFÍA

Las referencias aquí expuestas se han ordenado por apellido de autor, o, en su caso, título, y han sido citadas convenientemente a lo largo de la presente memoria. En el caso de referencias en línea, se incluye además la fecha de la última comprobación sobre su validez.

- Aracil, Alfredo. “Juego y Artificio. Autómatas y otras ficciones en la cultura del Renacimiento y la Ilustración”, Ed. Cátedra, 1998
- Bohn, James. “ILLIAC I”, http://ems.music.uiuc.edu/history/illiac.html [visitado el 25/05/2010]
- Brown, Andrew. Sorensen, Andrew. “JMusic: Music Composition in Java” http://jmusic.ci.qut.edu.au/ [visitado el 25/05/2010]
- Bush, Douglas E., Kassel, Richard “Encyclopedia of keyboard instruments: the organ”, Ed. Routledge, 2006
- Diccionario de la Real Academia de la Lengua Española http://www.rae.es [visitado el 25/05/2010]
- Drools Expert User Guide, http://www.jboss.org/drools [visitado el 25/05/2010]
- Encyclopedia Britannica, http://www.britannica.com [visitado el 25/05/2010]
- Friedman-Hill, Ernest. “Jess, the rule engine for the Java Platform”, 2008, http://www.jessrules.net [visitado el 25/05/2010]
- Gómez-Zamalloa Gil, Miguel. “Sistema de composición musical automática. Aproximaciones preliminares” http://gaia.fdi.ucm.es/people/pedro/aad/miguel_zamalloa.pdf [visitado el 25/05/2010]
- Jordá, Sergi. “Música e Inteligencia artificial” http://www.iua.upf.es/~sergi/musicia.htm [visitado el 15/02/2010]

- Koster, Jan. “J.S. Bach Home Page, Biography, Portaits and Literature” http://www.jsbach.org/biography.html [visitado el 25/05/2010]
- Jesús Lara Popoca. “Al son de las matemáticas Música y Números”, 2009 http://personales.ya.com/casanchi/rec/alsonmatematica01.htm [visitado el 25/05/2010]
- Pérez Ortiz, Juan Antonio. “Música Fractal: El sonido en el caos”, 2000, http://www.sectormatematica.cl/fractales/musicaenelcaos.pdf [visitado el 25/05/2010]
- Riley, Gary. “what is clips?”, 2008, http://clipsrules.sourceforge.net [visitado el 25/05/2010]
- Ríos, Rafael. “Entre las matemáticas y la música”, 2004, http://www.uv.es/metode/anuario2004/65_2004.htm [visitado el 25/05/2010]
- Robles, Luis. “Designing Music” http://www.designingmusic.org [visitado el 25/05/2010c], “Armonía” [visitado el 25/05/2010a], “Análisis” http://www.haciendomusica.com [visitado el 25/05/2010b]
- Samper Márquez, Juan José. “Introducción a los sistemas expertos” http://www.redcientifica.com/doc/doc199908210001.html [visitado el 25/05/2010]
- Sandred, Örjan. Laurson, Mikael. Kuuskankare, Mica. “Revisiting the Illiac Suit – a rule based approach to stochastic processes” http://www.sandred.com/texts/Revisiting_the_Illiac_Suite.pdf [visitado el 25/05/2010]
- Tiburcio Solís, Susana. “Teoría de la Probabilidad en la Composición Musical Contemporánea” http://sectormatematica.cl/musica/Tesis%20musica%20y%20matematicas%20Mozart.pdf [visitado el 25/05/2010]
- Zamacois, Joaquín. “Tratado de armonía. Libro I”, Ed. Spanpress Universitaria, 1945
- “Estructuras del lenguaje musical” http://www4.ujaen.es/~imayala/_private/estructuras/TEMA%204%20armonia.pdf

[visitado el 25/05/2010]

- “Finale” http://www.finalemusic.com [visitado el 25/05/2010]
- “6. Intervalos, escalas y tonalidad” http://aam.blogcindario.com/ficheros/art13.pdf [visitado el 25/05/2010]
- “Juego de los dados de Mozart” http://www.dpye.iimas.unam.mx/mozart/2.html [visitado el 25/05/2010]
- “Rete Algorithm” http://www.redhat.com/docs/en-US/JBoss_SOA_Platform/4.2.CP05/html/JBoss_Rules_Reference_Manual/sect-JBoss_Rules_Reference_Manual-Rete_Algorithm.html [visitado el 25/05/2010]
- “Sistemas Expertos” http://www.monografias.com/trabajos10/exper/exper.shtml [visitado el 25/05/2010]
- “Teoría Musical” http://www.aprende-gratis.com/teoria-musical [visitado el 25/05/2010]
- “Tutorial: The technology of MIDI” http://www.midi.org/aboutmidi/tut_techmidi.php [visitado el25/05/2010]