
Facultad de Informática
Departamento de Computación

# Herramienta para armonización musical mediante Answer Set Programming

**Autores:** Martín Prieto, Rodrigo
Cabalar Fernández, Pedro

*A Coruña, a 1 de junio de 2015*

# Índice general

**Página**

# Capítulo 1
# Introducción

## 1.1. Sobre el proyecto

El proyecto consiste en la construcción de una herramienta software capaz de tomar como entrada una partitura polifónica (es decir, de múltiples voces) parcial y completarla respetando las reglas básicas de armonía, de modo similar a los ejercicios habituales en esta disciplina musical. Se intentará abarcar la estructura musical de forma horizontal (compases) y vertical (múltiples voces) pero, a priori, se obviarán tanto posibles variaciones del ritmo como la incorporación de distintos estilos musicales. Este problema de partida se puede especificar en términos de resolución de restricciones, un campo para el que existen distintos formalismos y herramientas disponibles. En concreto, el proyecto usará el paradigma de *Answer Set Programming* una variante de Programación Lógica de uso frecuente para la Representación del Conocimiento y la resolución de problemas. La principal ventaja de ASP para este caso es la facilidad que otorga el uso de predicados simples a la hora de definir ciertos sucesos dentro de la partitura así como añadir directamente las reglas usadas en armonía bajo la forma de reglas de programación lógica. Esto proporciona una enorme flexibilidad, ya que ASP es un paradigma totalmente declarativo, en el que sólo se realiza la especificación del problema, y no se describe el método de resolución que se aplica para el mismo. Otra ventaja importante de ASP para este escenario es la posibilidad de usar preferencias, ya que algunas reglas armónicas no son estrictas, sino que se busca que se respeten en la medida de lo posible. Existen además antecedentes de uso de ASP para composición musical. En concreto, el sistema ANTON permite también la composición polifónica (mayormente, dos voces) siguiendo el estilo musical de

reglas "Palestrina"de la música renacentista. Esta herramienta es más elaborada que la presente propuesta, ya que realiza la composición completa de una pieza, incluyendo figuras rítmicas complejas. La principal diferencia es que el presente anteproyecto está orientado a la armonización en piezas de cuatro o más voces, como las habituales en música coral.

### 1.1.1. Módulos

#### 1.1.1.1. Entrada y preprocesado

Mediante el uso de una herramienta de edición musical, se permitirá al usuario introducir la partitura a completar. Dicho editor o interfaz producirá ficheros en formato MusicXML para, más adelante, convertir las notas de este fichero a hechos en ASP que puedan ser interpretados. La conversión de MusicXML a hechos lógicos se realizará mediante el diseño e implementación de un *parser* específico para el problema.

#### 1.1.1.2. Procesado

El núcleo lógico del programa. Aquí es donde se realizará la armonización vía ASP, aplicando preferencias indicadas y produciendo una salida en el formato adecuado, correspondiendo con el de entrada. Como ya se ha mencionado, la idea es crear predicados ASP que permitan comprobar la armonía de forma horizontal a lo largo de los compases y en vertical, teniendo en cuenta qué sucede en el resto de voces para que el programa "comprenda" qué tonalidad se está desarrollando en el compás actual, hacia donde fluye la melodía, etc. Principalmente se definirán predicados que "rellenan" la partitura de forma arbitraria para luego añadir restricciones que prohíben soluciones no deseadas, evitando disonancias, saltos de quinta, acordes tritono y demás figuras incorrectas en lo que a la música clásica se refiere o resultan poco agradables al oído. También se especificarán preferencias que describen reglas no estrictas que se deberán satisfacer en la medida de lo posible, de modo que ASP proporcionará soluciones que optimicen ese conjunto de preferencias. Tras hallar la solución, se producirá un nuevo fichero MusicXML con las correcciones necesarias.

#### 1.1.1.3. Postprocesado y Salida

Finalmente, y de forma opcional, se contempla la posibilidad de incluir un módulo que tome el fichero MusicXML corregido y que enriquezca el resultado. Este módulo no será desarrollado en ASP sino que tomará patrones y los aplicará sobre la partitura para darle una sonoridad más limpia y cuidada (división de figuras, ligadura de notas largas iguales consecutivas, cambios de estilo, instrumentación, etc). Finalmente, la salida se producirá en el formato deseado por el usuario, siendo los más interesantes la partitura en PDF, formatos compatibles con el lenguaje de notación musical Lilypond o inclusive en formato MIDI para su reproducción sonora.

## 1.2. Motivación

La Inteligencia Artificial estudia como crear sistemas que se comporten de manera inteligente, es decir, que sean capaces de razonar, deducir y resolver problemas del mismo modo que pudiera hacerlo una persona. Se busca que estos sistemas sean autónomos, se busca que estos sistemas sepan justificar sus resultados, se busca que estos sistemas sean capaces de aprender para poder desempeñar su tarea con mejores resultados o en menos tiempo. Pero la inteligencia conlleva más cosas, tales como la creatividad. La creatividad es el impulso por el cual alguien (o algo) decide crear una obra de la nada. No solo una obra artística, también obras funcionales como lo fueron los grandes inventos del pasado.

Existe, no obstante, controversia con respecto al campo, como suele suceder siempre que un ordenador empieza a conseguir hacer lo que antes solo podían hacer las personas. En el caso del sistema Emily Howell, por ejemplo, ha habido numerosos directores de orquestas que se han negado a interpretar sus composiciones al no provenir de un compositor humano. Existe, a ojos de los más conservadores respecto al tema, el miedo a que el esfuerzo de la composición musical pierda su significado. Si no podemos distinguir además qué piezas han sido compuestas por máquinas y cuales no, el problema se acentúa.

En este caso, la motivación del trabajo es una mezcla entre creatividad y la necesidad de solucionar un problema. Se habla de creatividad porque el proyecto está aplicado a un campo inherentemente creativo, como es la música. Pero al mismo tiempo, pretende ser una herramienta que ayude a estudiantes de música a

progresar en su trabajo. Este sistema inteligente, será capaz de razonar, deducir, y último lugar crear, la armonía de piezas musicales sencillas. Si bien esto no es un trabajo completo de composición, sí que debería ayudar a corregir partituras allí donde el sistema detecte incoherencias con la armonía creada o ya presente en la pieza.

El interés por la aplicación del *Answer Set Programming* en la música derivó de un trabajo previo con el Profesor Cabalar durante el curso de la asignatura de Representación del Conocimiento y Razonamiento Automático donde los alumnos construimos un sistema que dada una melodía, producía un canon polifónico. El presente trabajo se planteó como una extensión generalista del problema, orientado a composición completa, aunque fue reducido para poder entrar en los límites de un trabajo de fin de grado debido a la gran complejidad de crear un sistema así. No obstante, la lógica proposicional es idónea para esta tarea ya que el conjunto de reglas de la armonía clásica usada en los niveles más elementales del Conservatorio no ha cambiado desde los orígenes de la materia. Es un conjunto de reglas conciso, no muy grande y más o menos estricto. Simplemente traduciendo este conjunto de reglas a restricciones del lenguaje de lógica proposicional y siendo capaces de extraer los hechos lógicos de una partitura, el sistema debería ser capaz de detectar los errores de la misma y solucionarlos, así como rellenar huecos dejados a propósito en la partitura o completar otras líneas melódicas para formar, por fin, la armonía de la canción.

Con fines académicos, el sistema podrá ayudar a estudiantes y profesores de armonía básica por igual, al poder corregir ejercicios de esta materia de modo rápido, incluso proponiendo soluciones no contempladas inicialmente. También ayudará en trabajos de composición armónica, explorando, de forma creativa, nuevas posibilidades para el autor de la pieza original. No solo es una propuesta interesante a nivel de investigación informática, sino también a nivel musical. Por último, pero no menos importante, cada vez más alumnos cursan estudios musicales de algún tipo desde edades más tempranas, ya sea por cuenta propia o a través de academias y conservatorios. El estudio del contexto ha revelado que si bien existe investigación con respecto a la computación musical, no hay muchos sistemas inteligentes finales dedicados a la composición o a la asistencia en la composición musical.

# Capítulo 2
# Contexto

## 2.1. Contextualización musical

El presente proyecto, pese a estar realizado para Ingeniería Informática, posee una importante carga de teoría musical. Pese a ser un documento técnico, los lectores del proyecto podrían no conocer en profundidad los conceptos musicales empleados en el documento. Debido a ello, se cree importante hacer una introducción a modo de glosario y punto de consulta, partiendo de los elementos más básicos de la teoría musical y llegar hasta aquellas reglas más complejas utilizadas para el desarrollo del proyecto.

### 2.1.1. Ritmo y Figuras

La parte más elemental de la música es la estructura rítmica de la pieza, es decir, los patrones de golpes sonoros que están presentes en la canción y que normalmente se repiten a lo largo de toda ella. Para componer estas secuencias rítmicas se utilizan una serie de figuras que representan sonidos de una duración relativa al *tempo* de la pieza, en combinación con otra serie de figuras que representan la ausencia de sonido durante esas mismas duraciones. Las primeras son denominadas figuras y las segundas, silencios.

La figura base en la música es la redonda, y mediante un proceso de escisión binaria se obtienen el resto de figuras presentes en las partituras. Una redonda equivale a dos blancas, una blanca equivale a dos negras, una negra equivale a dos corcheas, una corchea equivale a dos semicorcheas... etc. A modo de detalle, para facilitar la lectura de figuras fraccionarias pequeñas, estas pueden ser unidas

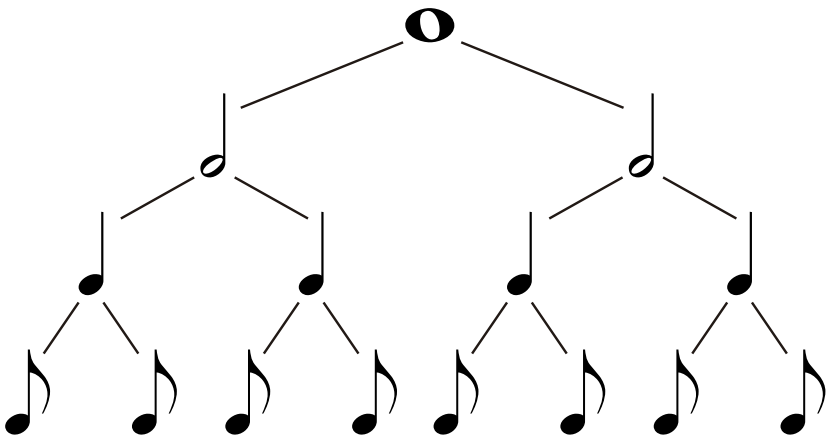

Figura 2.1: Diagrama de subdivisión de figuras

mediante una barra horizontal si aparecen de forma consecutiva. Es importante tener en cuenta que aunque la duración total de dos figuras iguales sea equivalente a otra figura de mayor duración, rítmicamente sí existe diferencia, no se debe olvidar en ningún momento que cada figura es un golpe de sonido con una duración determinada y por tanto, pese a que en cuanto a duración una redonda equivale a cuatro negras, a la hora de interpretar la pieza, una redonda será un único sonido largo mientras que cuatro negras serán cuatro sonidos más cortos. Esto en combinación con los silencios, componen los patrones de ritmo de los que se hablaba anteriormente.

Dado que en inglés estas figuras se llaman mediante el nombre de la fracción de redonda que representan también pueden ser representadas mediante el número del denominador de dicha fracción. De este modo la redonda sería 1, la blanca 2, la negra 4, la corchea 8, la semicorchea 16,etc. A mayores existen modificadores de la duración de las figuras como el puntillo, un pequeño punto situado a la derecha de las figuras que alarga la duración de las mismas durante la mitad de la duración de la figura a la que acompaña.

El *tempo*, anteriormente mencionado, es una relación que indica la cantidad aproximada de figuras de negra que suenan en un minuto, debido a eso la magnitud que representa el *tempo* se denomina BPM o *Blacks per Minute*.

La partitura está dividida en compases, y al principio de la pieza se indica el tipo de compás de la misma. Se representa con dos números colocados en vertical

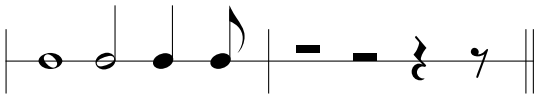

Figura 2.2: Figuras con sus respectivos silencios, de izquierda a derecha: Redonda, blanca, negra, corchea

que indican respectivamente la cantidad de figuras que caben en cada compás y el tipo de dicha figura, utilizando el sistema del numerador de la fracción de redonda detallado anteriormente. A modo de aclaración, el tipo de figura utilizado en el compás solamente indica la figura base del mismo, pero eso no impide que se utilicen fracciones o figuras de mayor longitud equivalentes siempre y cuando se respete que la suma de sus duración no supere la especificada en el tipo de compás.

Según la cantidad de figuras que caben en cada compás, estos reciben diferentes nombres. En la música moderna lo más habitual es encontrar compases cuaternarios, aunque en algunas corrientes de la música clásica como el *Waltz* predominan los ternarios.

Figura 2.3: Ejemplos de tipos de compases: Cuaternario y Ternario

El acento en la música es un mero hincapié interpretativo en los tiempos fuertes de la pieza. Aunque puede existir una acentuación arbitraria a lo largo de la partitura, los compases determinan además los tiempos fuertes y normalmente acentuados. Dependiendo del tipo de compás estos varían, aunque la primera figura de cada compás suele estar acentuada y ser considerada fuerte. No obstante, la diferenciación de tiempos débiles y fuertes es importante en la melodía y por tanto en la armonía, por ello se revisará este concepto en los puntos siguientes.

### 2.1.2. Melodía y pentagrama

La melodía de la pieza es la secuencia de notas que siguen las diferentes voces a lo largo de la partitura. Para ello se utilizan siete notas fundamentales diferentes:

Do, Re, Mi, Fa, Sol, La y Si (C, D, E, F, G, A, B en la notación internacional). Entre cada uno de estas notas fundamentales se establece una distancia de un tono, excepto entre Si y Do; y entre Mi y Fa, donde solo hay medio tono (también denominado semitono). Las fundamentales pueden ser alteradas mediante modificaciones conocidas como Sostenido y Bemol, que suben o bajan, respectivamente, un semitono a dichas notas, por eso podemos entender que contamos, en total, con 12 sonidos distintos.

Por este motivo no se contemplan notas como Si sostenido o Mi bemol, ya que en realidad serían Do y Fa respectivamente. Estas siete notas fundamentales (o doce sonidos), tienen asociadas a su vez una octava. Cada octava es un conjunto formado por esos mismos sonidos pero en un tono más agudo o más grave. Si bien esto produce un sonido que es diferente, el mismo sonido de diferentes octavas es equivalente en términos de armonía.

La representación de estos sonidos se hace sobre un pentagrama, bandas formadas por cinco líneas con sus respectivos cuatro huecos donde pueden colocarse las notas de nuestra partitura, representando cada línea y hueco consecutivos una nota fundamental diferente. Ya que esto solo permitiría representar nueve notas, se pueden utilizar líneas y huecos adicionales situados encima y debajo del pentagrama para indicar notas más graves o más agudas. En conjunción con las alteraciones ya mencionadas (Sostenido y Bemol) en un mismo pentagrama pueden representarse más de una veintena de sonidos diferentes.

Para poder interpretar el sonido correspondiente a cada línea o hueco existe la clave. La clave es un símbolo situado al principio de cada pentagrama e indica la nota correspondiente a una de las cinco líneas. El resto se calculan subiendo o bajando huecos y renglones a partir de esa línea.

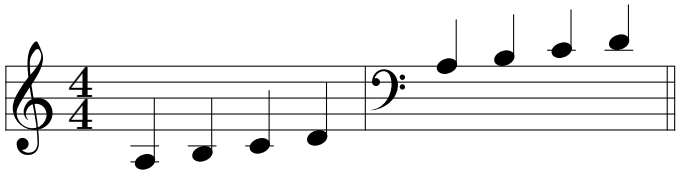

Figura 2.4: La misma secuencia de notas en clave de Sol y de Fa

En la Figura 2.4, la clave de Sol indica que en la segunda línea, contando desde abajo, estaría ubicada la nota sol, mientras que la clave de Fa indica que en la cuarta línea, contando también desde abajo, se ubicaría el sonido Fa. La

finalidad de las diferentes claves es ofrecer diferentes puntos de referencia, ya que no solo indican nota sino también octava, evitando así sobrecargar la partitura con notas fuera de las cinco líneas principales. La clave de Sol utiliza sonidos más agudos, mientras que la de Fa utiliza sonidos más graves. Es común encontrar, por ejemplo, partituras para piano donde la mano izquierda estará escrita en clave de Fa al ser notas más graves y en clave de Sol la de la mano derecha, al ser notas más agudas.

La transposición es un proceso mediante el cual una nota es desplazada una cantidad de semitonos hacia arriba o hacia abajo. Aplicado a una sección de una pieza musical, todas las notas de dicha sección subirían o bajarían la misma cantidad de semitonos especificada.

Un salto melódico es la diferencia de frecuencia entre dos sonidos consecutivos de una misma melodía. De este modo, si el siguiente sonido es más agudo se producirá un salto ascendente, y si es más grave, descendente. Las secuencias ascendentes y descendentes se combinan con el ritmo de la canción para crear tramos de tensión melódica, de resolución, de reposo... Para entender estos conceptos es necesario introducir el concepto de escala.

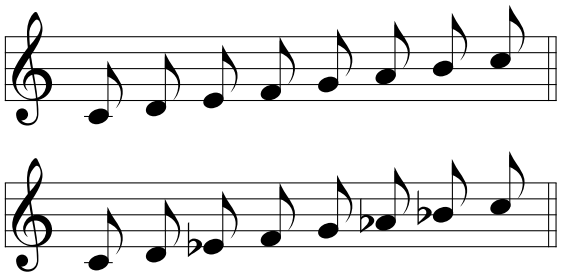

Figura 2.5: Escalas de Do Mayor y Do Menor

Una escala no es más que un subconjunto de los doce sonidos disponibles donde sus miembros cumplen una distribución concreta de separaciones tonales. La escala más conocida es la de Do Mayor, ya que empezando desde Do, pasa por las siete notas fundamentales, pero si se vede desde la distribución de tonos y semitonos entre sus notas (Tono, Tono, Semitono, Tono, Tono, Tono, Semitono) se puede construir cualquier otra escala mayor dada una nota arbitraria de partida. Las escalas pueden ser, según su distribución tonal Mayores o Menores, así como recibir diferentes nombres según la cantidad de sonidos que incluyan. La escala de

Do Mayor es en realidad la escala Heptatónica de Do Mayor ya que incluye siete sonidos, pero otras escalas como las Pentatónicas también están ampliamente utilizadas en la música moderna.

En aquellas piezas en las que se trabaja sobre una determinada escala, es más cómodo especificar una armadura. La armadura es un conjunto de alteraciones al principio del pentagrama, justo a la derecha de la clave, que indica qué alteraciones tendrán las notas de ese pentagrama a lo largo de toda la canción o hasta que se especifique lo contrario. De este modo, para la escala de Do Menor de la Figura 2.1.2 podría especificarse una armadura que indicase tres bemoles en las líneas de mi, la y si. Si quisiéramos escribir esas notas sin la alteración, para contrarrestar la armadura, habría que usar otra alteración denominada becuadro. El becuadro, también conocido como natural, elimina las alteraciones establecidas por la armadura del pentagrama.

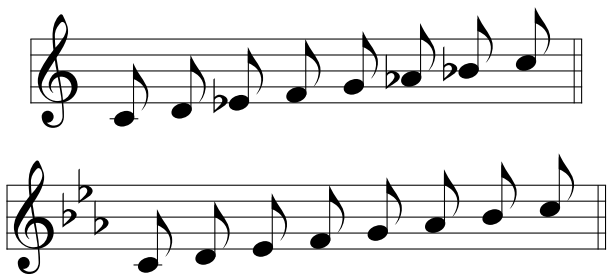

Figura 2.6: Escala de Do Menor sin armadura y con armadura

### 2.1.3. Armonía

Si la melodía es la sucesión de notas en el tiempo, se puede entender la armonía como los conjuntos de notas que suenan a la vez a lo largo de un intervalo de tiempo. Por eso se habla de que el análisis melódico es horizontal y el armónico vertical, aunque esto no es del todo cierto, pues al fin y al cabo la armonía analiza también intervalos horizontales de tiempo (normalmente acotados en compases) y la estructura general de la pieza. Para poder estudiar o crear armonía se debe contar siempre o bien con varias voces o bien con instrumentos polifónicos capaces de producir sonidos simultáneos (Como el Piano o la Guitarra)

Uno de los conceptos más importantes dentro de la armonía es la tonalidad. La tonalidad define siete grados para una tonalidad, y están relacionados con la

escala de dicha tonalidad. Cada grado está referido a cada una de las notas de las calas y se representa mediante un número romano de forma consecutiva, además de un nombre. Los grados más relevantes son el I, IV y V, mientras que los menos importantes o débiles son el II y el VI. A mayores, los grados muy débiles son el III y el VII.

- I (Tónica)
- II (Supertónica)
- III (Mediante)
- IV (Subdominante)
- V (Dominante)
- VI (Superdominante/Submediante)
- VII (Sensible)

Según la teoría, un acorde son tres o más sonidos simultáneos a una distancia de tercera entre sí de forma ascendente. Analizando el tercer grado de la escala de las notas del acorde, podemos determinar si este es mayor o menor. La nota generadora del acorde suele ser la tónica y además la más grave del mismo, si no es así se dice que el acorde está invertido.

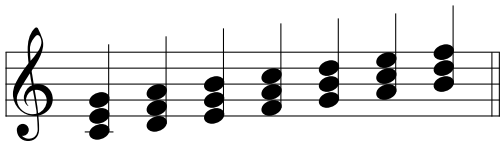

Figura 2.7: Acordes de la tonalidad de Do Mayor

En una escala de modo mayor, el I, IV y V grado son acordes perfectos mayores, el II, III y VI acordes perfectos menores, y el VII un acorde disminuido. En una escala de modo menor, el I y IV grado son acordes perfectos menores, el II y VII son acordes disminuidos, el III es aumentado.

Ciertas combinaciones de acordes producen sensación de tensión, mientras que otras producen sensación de reposo. Algunos acordes, según el contexto, tienen

un sentido conclusivo y otros, un sentido transitorio. Estos conceptos son muy importantes en composición, así como en armonización.

La armonización es el proceso de construir una armonía mediante varias voces que, sin modificar la melodía de la pieza, enriquezca la misma reforzando la tonalidad en cada instante. Esto se consigue identificando la tonalidad de la pieza en cada tramo, tomando las notas del mismo y averiguando la escala sobre la que se ha construido la melodía. Dada la escala podemos identificar la tonalidad, y con la tonalidad construir por fin los acordes.

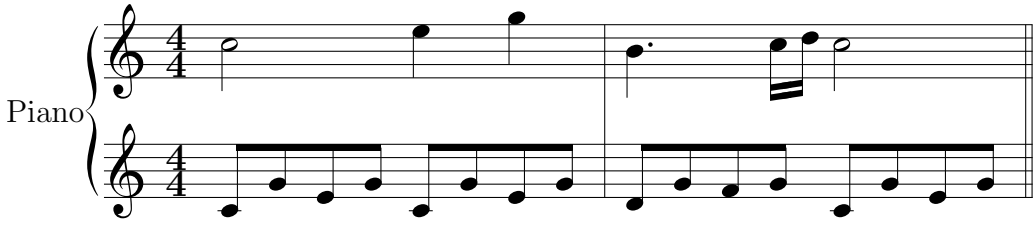


Figura 2.8: Ejemplo de polifonía para Piano (Sonata en Do Mayor, Mozart)

No obstante, existen reglas a respetar, ya que no es tan sencillo como tomar cada nota de la melodía como nota generadora de un acorde y construir el acorde mayor o menor correspondiente a la misma. Por otra parte no basta con tocar el acorde principal de la tonalidad a lo largo de toda la pieza, ya que si bien esto no sería erróneo, podría producir problemas con las fluctuaciones de la melodía, y no cumpliría, en último extremo, la finalidad de enriquecer la melodía.

Los acordes no solo deben respetar la melodía, sino que además deben ser escogidos de tal forma que se enlacen correctamente, normalmente debido a las notas compartidas entre dos acordes consecutivos. Las notas distintas al enlazar dos acordes pueden producir sonidos indeseados con respecto a la melodía o causar saltos melódicos no agradables al oído. Para evitar estos problemas se estudia a su vez, horizontalmente, las nuevas líneas melódicas formadas por las diferentes voces de los acordes. Los diferentes movimientos de estas voces podrían ser, entre otros:

- Paralelo: Dos voces de varios acordes siguiendo una distancia, realizando los mismos saltos melódicos.
- Oblicuo: Una voz representada con una nota de larga duración y otra voz moviéndose libremente.

- Directo: Dos voces moviéndose a la vez de forma ascendente o descendente, pero no en paralelo.
- Contrario: Dos voces moviéndose en sentidos distintos.

En movimientos como el paralelo o el contrario pueden surgir problemas armónicos, como formar un intervalo de octava o quinta justa sobre las mismas voces. El movimiento directo también presenta problemas armónicos, en voces extremas (Bajo y Soprano), si la voz de la Soprano (más aguda) no se mueve por grados conjuntos, se presenta ese problema. En partes intermedias (voces centrales o una voz central y otra extrema), si una de esas dos voces no se mueve por grados conjuntos, ese enlace sería incorrecto. En voces seguidas, se busca evitar los saltos de octava

## 2.2. Contexto Tecnológico

La mayor parte del trabajo en la creación y composición musical en ordenadores se ha extraído de la relación entre la teoría musical y las matemáticas. No es difícil concluir que la música y sus reglas son fácilmente modelables de forma matemática.

Dentro de la música en computación existen varias ramas diferenciadas, aunque en el contexto de un mismo trabajo pueden verse mezcladas más de una. Hablamos de composición asistida y de de sistemas inteligentes orientados a composición y, aunque se tratarán con más detalle en los siguientes puntos, todos ellos, así como el software desarrollado en dichos campos, están destinados a la creación y composición musical. Se detallan, además, algunos de los formatos más comunes de representación musical.

Si bien el sistema planteado en el proyecto no es un compositor, se enmarca dentro de este mismo contexto, y por tanto es necesario desglosarlo para entender en qué punto se encuentra la tecnología desarrollada en el momento de la publicación de este documento.

### 2.2.1. Formatos

Debido al contexto en el que se enmarca este proyecto no se cree necesario considerar formatos de salida finales, como OGG, WAV o MP3 ya que como su nombre indica, son formatos utilizados solo para reproducción que no permiten extraer ni editar información musical de forma precisa. Así mismo tampoco se contemplan formatos de representación de partituras en forma de imágenes como SVG, PNG o PDF por motivos similares.

#### 2.2.1.1. MusicXML

MusicXML, MXML o *Music Extensible Markup Language* es una extensión del formato XML usado en la representación de música occidental. No solo contiene información musical sino que también incluye información de su representación en papel, tal como márgenes, tamaños de fuente, posición de las notas en coordenadas, etc. Hace uso del sistema de tags anidados de XML para agrupar los diferentes bloques de información de una pieza, como las voces, los compases o

la información individual de cada nota. Es un formato muy rico aunque difícil de escribir correctamente a mano, es por esto que normalmente se usa solo como formato de intercambio entre software que lo aceptan como entrada y salida.

#### 2.2.1.2. LilyPond

LilyPond es un conjunto formado por el software y el formato de fichero homónimos. LilyPond como formato usa su propio lenguaje de marcado, los tags de LilyPond se parecen más a los usados en LaTeX. De forma similar a MusicXML, incluye información de representación final, aunque en mucha menor cantidad (Sólo tamaño de papel, márgenes o sangrados). La información musical de la canción está anidada por secciones de forma similar a MusicXML, aunque ésta se organiza de forma mucho más intuitiva para el lector humano del fichero. Es un formato ligero pensado para poder ser editado a mano, aunque la mayoría del software musical actual lo soporta como entrada y salida.

#### 2.2.1.3. MIDI

MIDI son las siglas de *Musical Instrument Digital Interface*. Es un standard técnico compuesto de un protocolo, una interfaz digital y conectores que permiten a una gran variedad de instrumentos electrónicos, ordenadores y otros dispositivos conectarse y comunicarse entre sí, principalmente con fines musicales, pero no siempre.

MIDI transmite mensajes de eventos que especifican notación, tono y velocidad, aunque también incluye información de modificaciones sobre estos sonidos como volumen, *vibrato* y marcas de tiempo con fines de sincronización entre dispositivos.

Estos mensajes pueden ser codificados en ficheros para reproducción, edición o simplemente como formato de representación musical pudiendo ser editado posteriormente. Dado que no contiene información final de audio, MIDI supone una gran ventaja en cuanto a espacio en disco, aunque el sonido final depende del equipo que reproduzca el fichero.

### 2.2.2. Software

#### 2.2.2.1. Composición Asistida

Dentro de la composición asistida encontramos principalmente software de composición general en forma de editores de partituras que incorporan herramientas para ayudar al compositor en el proceso. Estas herramientas pueden ser corrección de la métrica de los compases, transposición de secciones de la canción, cambios de tonalidad, construcción de acordes dada una nota generadora, etc.

##### 2.2.2.1.1 MuseScore

MuseScore es un editor *WYSIWYG* capaz de exportar a varios formatos de representación musical digital, tales como MIDI, LilyPond o MusicXML.

##### 2.2.2.1.2 Sibelius

Sibelius permite trabajar con gran variedad de modos de entrada de notas para sus partituras, desde los formatos convencionales hasta a través de instrumentos con salida MIDI o mediante el escaneado de partituras en papel haciendo uso de OCR.

##### 2.2.2.1.3 Finale

Finale destaca por la cantidad de ajustes que permite realizar sobre el pentagrama a un nivel de detalle muy fino, aunque presenta una curva de aprendizaje muy elevada. Sus principales características tienen que ver con la visualización del pentagrama, ya que posee *plug-ins* que se encargan de que el espacio entre las notas sea el correcto o que no haya colisiones entre notas de diferentes voces entre otros.

#### 2.2.2.2. Sistemas Inteligentes

La música si bien puede modelarse de forma matemática, con reglas estrictas que derivan en algoritmos de composición, requiere creatividad. Un algoritmo determinista no puede ser creativo, ya que para la misma entrada, siempre producirá la misma salida. Si bien existe la composición algorítmica como aproximación a la música compuesta por ordenadores, no es relevante para este trabajo.

Dentro de la inteligencia artificial, se han realizado aproximaciones a la composición musical desde gran parte de las ramas principales del campo

#### 2.2.2.2.1 EMI y Emily Howell

Desarrollado por David Cope, EMI o Experiments in Music Composition es un sistema capaz de identificar el estilo presente en una partitura incompleta y completar la cantidad de notas restantes que el compositor requiera. El trabajo de Cope estudia la posibilidad de emplear gramáticas y diccionarios en la composición musical. EMI derivó en el software Emily Howell. Emily Howell utiliza EMI para crear y actualizar su base de datos, pero cuenta con una interfaz a través de la cual se puede modificar, a través de *feedback*, la composición. Cope enriqueció y pulió Emily Howell con su propio estilo musical para crear varios discos que después fueron publicados.

#### 2.2.2.2.2 ANTON

ANTON es un sistema de composición rítmica, melódica y armónica basado en Answer Set Programming. ANTON compone breves piezas musicales desde cero o partiendo de partituras incompletas utilizando un estilo basado en el del compositor renacentista Giovanni Pierluigi da Palestrina. Recibe como entrada ficheros con las notas codificadas como hechos lógicos para después rellenar las secciones incompletas o añadir nuevas notas hasta que la pieza está completa. ANTON crea y completa dichas piezas teniendo en cuenta el número de tiempos rítmicos de las mismas y seleccionando la nota correspondiente de acuerdo a la nota o estado anterior

#### 2.2.2.2.3 Vox Populi

Vox Populi utiliza algoritmos evolutivos para componer música en tiempo real. En este sistema, se parte de una población de acordes codificados mediante el protocolo MIDI para después mutarlos y seleccionar los mejores acordes a criterios puramente físicos relevantes para la música. Su interfaz gráfica permite al usuario controlar la función de *fitness* del proceso evolutivo así como los atributos del sonido producido.

#### 2.2.2.2.4 CHORAL

CHORAL es un sistema experto que funciona como armonizador en el estilo clásico de Johann Sebastian Bach. Las reglas que utiliza el sistema representan conocimiento musical desde varios puntos de vista de la coral. El programa armoniza melodías corales mediante un sistema de generación y prueba con *backtracking*. La base de conocimiento de CHORAL permite realizar modulaciones propias del estilo, crear patrones rítmicos e impone restricciones complejas para mantener el interés melódico en las voces intermedias.

#### 2.2.2.2.5 CHASP

CHASP es una herramienta creada por el grupo Potassco para calcular progresiones de acordes mediante Answer Set Programming partiendo de cero, pudiendo especificar clave y duración. A diferencia del presente proyecto no toma un fichero de entrada para armonizar piezas, pero sí que es capaz de dotar a la salida del programa de diferentes estilos rítimicos.

# Capítulo 3

# Desarrollo

Se ha seguido un proceso de desarrollo en espiral con prototipado. El diseño y desarrollo de los diferentes componentes del sistema se presta a este tipo de ciclo ya que permite crear y probar en cada iteración un producto prototipo, centrándose en las primeras iteraciones en diseñar más e implementar menos y en las últimas solo refinar el software final.

No obstante dentro de este ciclo en espiral se han diferenciado varias etapas correspondientes a cada uno de los módulos del mismo.

## 3.1. Iteración 1

Se empezó por identificar los formatos, herramientas y componentes del proyecto. Como software de entrada para las partituras se escogió MuseScore, principalmente por ser *opensource* y por su sencillez. MuseScore no posee una curva de aprendizaje difícil, ya que la introducción de las notas se puede hacer de forma visual en su interfaz mediante el ratón o con el teclado y no requiere mayor preparación para crear y exportar una pieza musical sencilla. Su soporte para *plugins* también resulta interesante, ya que el proyecto podría llegar a transformarse en un *plugin* para la herramienta si se desease una mayor integración con la misma. MuseScore además ofrece soporte para los tres formatos de ficheros musicales contemplados, así como para PDF y otros formatos finales de imagen.

Para el formato de entrada y salida, se compararon las propiedades de MIDI, LilyPond y MusicXML. Los tres formatos ofrecen posibilidades de edición, aunque cada uno sirve a un propósito diferente. MIDI no trabaja con ficheros textuales, sino que codifica de forma binaria toda la información de los even-

tos de la canción, LilyPond es un formato de texto plano pensado para que una persona pueda editarlo a mano, posee una estructura de marcado a través de etiquetad de solo apertura o autocerradas y trabaja con identaciones para formar la jerarquía del fichero, esto requiere al usuario escribir mucho menos, pero puede suponer problemas para un *parser* convencional debido a que la identación de LilyPond no está estandarizada. Por último MusicXML se presenta como el formato idóneo para la tarea, ya que al ser una extensión de XML, está orientado a que una máquina pueda leerlo y crear una estructura en memoria con tod ala información que necesita para poder extraer los datos de la partitura. Además, la implementación de un *parser* de un lenguaje etiquetado como XML es un problema convencional y fácilmente abarcable.

En cuanto a la tecnología escogida para diseñar y construir el *parser* se optó por las bibliotecas Flex y Bison para C.

Analizando la especificación del esquema de MusicXML de cara a desarrollar el *parser* se identificaron las diferentes partes del mismo. MusicXML incluye información tanto musical como de representación gráfica de los diferentes elementos de la partitura. Todos esta información gráfica es generada automáticamente por el software que exporta el fichero MusicXML (MuseScore para el caso) y resulta irrelevante a la hora de extraer los hechos lógicos presentes en la partitura, por tanto se decidió obviarla.

MusicXML declara inicialmente el número de voces presente en la partitura mediante la etiqueta part-list y sus etiquetas anidadas score-part. Más adelante, las etiquetas part se encargan de contener los compases mediante la etiqueta measure que a su vez contienen las etiquetas note. Son estas últimas etiquetas las que hay que desglosar para extraer los hechos lógicos, aunque la información de las etiquetas measure también es relevante, así como poder asignar un identificador a cada voz de la partitura para poder diferenciarlas a nivel lógico.

```
<note default-x="74.65" default-y="-25.00">
    <pitch>
        <step>A</step>
        <octave>4</octave>
    </pitch>
    <duration>1</duration>
    <voice>1</voice>
```

```
    <type>quarter</type>
    <stem>up</stem>
</note>
```

La etiqueta note posee dos parámetros visuales, default-x y default-y, que indican su posición en coordenadas x e y, como ya se ha comentado antes, esta información es meramente visual. La etiqueta pitch describe el sonido de la nota mediante el nombre de la nota usando notación internacional y la octava de la nota. Duration especifica la duración en tiempos del compás de la nota actual, voice asigna este sonido a la voz correspondiente, type indica la figura mediante el sistema fraccionario y stem dice si la plica de la nota es ascendente o descendente, de nuevo esto es meramente visual.

Ya que el armonizador solo contemplará partituras uniformes (es decir, con un solo tipo de figura presente), unicamente necesitamos saber la voz, el compás, el tiempo en el que ocurre la nota relativo al compás y el sonido de la nota.

Con esta información en mente se procedió a desarrollar el *parser* de MusicXML a hechos en ASP. Se hizo uso de Flex para el análisis léxico y de Bison para el análisis sintáctico. El primer paso fue identificar los diferentes tags relevantes para el análisis léxico, estos son note para las notas, step para el ritmo, rest para los silencios, chord para los acordes, octave para la octava de cada nota, alteration para identificar sostenidos y bemoles, part para las diferentes voces de la pieza y por último elementos relevantes para el análisis de XML pero carentes de significado real para los hechos lógicos de la partitura tales como la apertura y cierre de etiquetas, campos textuales o símbolos varios.

En el código de Bison se diseñó la gramática primero y se integró junto con el código C encargado de tomar los datos extraídos por el *parser* y exportarlos a un fichero de hechos lógicos listo para ser interpretado. La gramática diseñada para ello parte de una regla inicial que puede derivar en un bloque compuesto por etiquetas anidadas o en un error si el fichero no tiene el formato y estructura adecuados.

Los bloques de etiquetas se descompusieron en tokens según su contenido o en dos partes (part1 y part2) dependiendo de si la etiqueta es autocerrada o no. Los errores que se pueden encontrar en este punto son que la etiqueta se abra pero no se cierre o que haya elementos de más no reconocidos tras la segunda parte de la misma.

Las dos partes de cada etiqueta no autocerrada se corresponden respectivamente con la apertura de la etiqueta (part1) y con el contenido y cierre de la misma (part2).

Por último se hace uso de una regla recursiva que permite anidar los bloques de etiquetas y contenido.

Las pruebas realizadas al *parser* revelaron que existía un problema de análisis al no poder verificar de forma sencilla que cada etiqueta se cerraba de modo correcto, es decir, que el nombre de la etiqueta que cierra un bloque sea el mismo del que la abrió, se implementó una pila en C para esta tarea. La implementación se realizó de modo que la pila no tuviese un tipo definido de partida, por flexibilidad, mediante el uso de punteros a void y el *typecast* de los mismos en ejecución, aunque en el caso del *parser*, solo se usó el tipo char*.

La implementación de la misma se hace mediante estructuras típicas de C enlazadas mediante punteros unidireccionales y el acceso a los datos de la pila se realiza mediante los conocidos métodos para operar con este tipo de datos (new, is empty, push, pop, etc.)

Además de la adición de la estructura de pila para verificar esto se incluyeron más opciones de error en varias de las reglas gramaticales de modo que resultase más fácil la depuración del *parser*. Esto reveló que había fallos en ejecución, ya que los resultados no eran los esperados. Este problema se debía a que la recursión se realizaba inicialmente mediante la posibilidad de que block derivase en otro block, así que se descompuso esta regla en un nuevo elemento body que añadía un paso más y abstraía los bloques de etiquetas y contenido pero a un nivel de granularidad algo más pequeño que body en sí. Tras la inclusión de este paso intermedio, el *parser* aún no terminaba de comportarse como era esperado y hubo que incluir reglas específicas para comprobar tags con formatos especiales tales como <?xml version> o <!DOCTYPE>. Por último se comprobó que el *parser* no contemplaba inicialmente tags autocerrados en su forma <tag></tag> es decir, sin contenido. Al corregir estos detalles, cualquier pieza musical era transformada a hechos musicales a la perfección.

## 3.2. Iteración 2

El prototipo objetivo de esta iteración es una primera versión del módulo de procesado en ASP. Para ello se fijan una serie de restricciones que simplifican algunos problemas detectados durante el análisis. Inicialmente no se tendrán en cuenta las notas adyacentes en la partitura ni subdivisiones de las mismas, no obstante y de cara a futuras iteraciones, se establece una figura base como período de análisis horizontal de la melodía y una figura base de subdivisión, aunque para esta iteración ambas se establecerán a negra (*quarter note*) de modo que se asignará el acorde correspondiente a cada golpe rítmico de la partitura teniendo en cuenta las múltiples voces que la componen de manera vertical.

Para empezar se ha modificado el parser y se ha incluido una opción para subdividir la partitura de forma automática en base a la nota más breve de la partitura o forzar toda la partitura a un solo tipo de figura omitiendo aquellas figuras de menor duración, aunque esto lleve a resultados incorrectos musicalmente. La salida deseada y el formato de entrada para el módulo de procesado ASP usan el formato "note(voz, tono, tiempo)"siendo voz el número de la voz que interpreta la nota, tono un valor numérico asociado a la nota, no en frecuencia sino calculado como el número semitonos a partir de una nota base A0, es decir, el La más grave que puede interpretar MIDI; y tiempo la posición de la nota en la partitura.

Para el desarrollo del módulo de procesado ASP se sigue el proceso conocido como *Generate and Test*, muy habitual en el paradigma. Como su nombre indica, consta de dos partes: generación y prueba. En la generación se usan reglas que definen todas los resultados posibles del problema para despues, en la fase de prueba, restringirlos mediante reglas que prohíben total o parcialmente ciertas combinaciones al exponerse a hechos lógicos aportados por el *parser*. Esta metodología se usará desde el principio en cada iteración para desarrollar la parte en ASP correspondiente ya que aunque las reglas de generación permanecerán intactas, estas serán revisadas por si fuera necesario cambiarlas o añadir alguna regla nueva.

En esta primera aproximación, las reglas de generación establecen los posibles acordes que compondrán la solución y se realiza una asignación acorde-unidad rítmica (como ya se comentó, la unidad rítmica base inicialmente será la negra)

en base a las notas presentes para un instante dado en cada voz. Para realizar esto es necesario interpretar cual es el grado de cada nota presente en la escala de la pieza, (de nuevo, por restricciones iniciales se considera Do mayor) ya que de este modo podremos especificar una serie de reglas que abstraigan el hecho de qué nota suena en si y lo sustituyan por qué grados aparecen en cada momento dado de la partitura, pudiendo deducir así qué acorde es el más correcto de entre todas las posibilidades, a más notas presentes de la escala correcta, menos posibilidades, de aparecer notas no pertenecientes a la escala, no se podrá resolver el acorde y el programa se detendrá con un error.

La fórmula empleada para derivar los grados de la escala es $[(valor - base) mod 12]$ siendo valor el valor númerico asignado a la nota y base la nota de la que se parte para calcular los grados. Esta expresión resulta en los semitonos de distancia entre la nota base y la nota actual. La operación de módulo es necesaria para abstraer la octava a la que pertenece la nota, aunque más adelante esto pueda ser un hecho relevante. Con una comparación directa del valor obtenido con una distribución dada según el modo (Mayor o Menor) de la escala es posible asignar los grados correspondientes a cada nota. Por último se generan hechos que indican cuantas voces toman parte en la pieza, así como los tiempos en los que suena al menos una nota.

Se crearon dos ficheros a mayores que especifican los acordes, otorgándole un nombre en notación numérica romana y creando múltiples predicados que especifican qué grados de la escala pertenecen a dicho acorde. De este modo no es necesario especificar en ningún momento cuantas notas tendrán los acordes con los que se trabaje, y aunque principalmente se trabajará con acordes de tres notas, esto permitirá ampliar la complejidad en el futuro al incluir acordes de dos o cuatro notas. Teniendo en cuenta los posibles acordes, y mediante una restricción de cardinalidad, se generan todas las soluciones iniciales, haciendo que solo se asigne un acorde a cada tiempo en el que suena al menos una nota. A mayores se especifica una restricción de integridad que anula cualquier solución en la que, para un tiempo en el que esté presente una nota concreta, el acorde asignado en la solución comprobada no contenga esa nota.

A mayores se ha incluído como parte de esta iteración, el diseño e implementación de un pipeline en python que automatice el proceso de *parseo* y de procesado mediante el módulo ASP. Dicho pipeline incluye la posibilidad de alternar entre

los modos mayor y menor mediante una opción. En este primer prototipo no se han utilizado *wrappers* para trabajar con *clingo* directamente a través del pipeline, por ello se implementó una pequeña funcionalidad de interpretación de la salida del módulo ASP a un vector de soluciones. En futuras iteraciones, se añadirá más funcionalidad a dicho pipeline, como más opciones de entrada o diferentes representaciones de las soluciones ofrecidas por el módulo ASP.

Se han realizado pruebas individuales a los módulos *parser* y ASP, además se han realizado pruebas al pipeline creado, sirviendo de este modo de pruebas de integración de los diferentes módulos y de la herramienta al completo.

Para el *parser* se han probado múltiples ficheros de entrada así como las distintas opciones que contempla. No presenta problemas para ficheros MusicXML estándar generados por Musescore y programas similares, ante la ausencia de fichero, ficheros incompletos, ficheros XML que no sean MusicXML o ficheros que directamente no sean XML el *parser* detiene la ejecución y produce el mensaje de error adecuado en cada caso. En cuanto a las tres opciones que presenta esta primera versión, la opción de ayuda -h detiene la ejecución independientemente de la presencia de otros parámetros y muestra el modo de uso del módulo parser por sí solo, la opción para especificar el fichero de salida -o funciona correctamente en cualquier caso, excepto cuando el directorio especificado para almacenar el fichero de salida no existe. En la siguiente iteración se planteará la posibilidad de dejar esto de este modo intencionadamente o si forzar la creación de los directorios necesarios. Por último la opción de subdivisión -s, que para este primer prototipo ha de ser especificada manualmente o dejada en su valor por defecto funciona para cualquier valor. Esto no es del todo correcto ya que la subdivisión solo debería ser posible para potencias de 2, en el segundo prototipo se corregirá esto.

El módulo ASP no presenta una gran complejidad de prueba ya que solo depende del fichero de entrada. Si este no está presente no produce resultado alguno, y en caso de estarlo y no ser correcto, produce un resultado de insatisfacibilidad. En caso de existir alguna solución esta es correctamente devuelta por pantalla.

Por último, en las realizadas para el pipeline se analizaron las diferentes opciones y casos de entrada de la herramienta. Ante los casos erróneos donde el módulo *parser* falla y no genera fichero LP de salida, el pipeline no detiene la ejecución y llama al módulo ASP igualmente, aunque este falla al no tener un fichero de hechos lógicos con el que funcionar, no obstante el pipeline si que de-

bería comprobar esto y por tanto se corrigió. Con ficheros de entrada adecuados el pipeline realiza una ejecución completa y recoge los resultados ofrecidos por el módulo ASP, ofreciendo por pantalla una representación más amigable así como un resultado de satisfacibilidad. Las opciones de esta primera versión del pipeline solo fueron dos, una que permite cambiar el modo de la amronía entre mayor y menor (-m major—minor) y otra que permite cambiar el número máximo de soluciones ofrecidos por el módulo ASP (-n N). La primera funciona como es debido, restringiendo los dos posibles valores de la opción a los especificados y deteniendo la ejecución en caso de encontrar un valor incorrecto, mientras que la segunda restringe también correctamente los valores de N a únicamente enteros y no hace falta mayor comprobación ya que cualquier valor entero es válido como opción para el módulo ASP.

Este primer prototipo se completó en dos semanas, siendo los mayores problemas de la iteración la falta de soltura con el paradigma y el lenguaje, así como detalles técnicos de falta de librerías y software.

## 3.3. Iteración 3

El objetivo de esta iteración es completar el desarrollo de un segundo prototipo de la herramienta que incluya las siguientes mejoras: Subdivisión real y automatizada de las notas de la pieza a la longitud de la nota mínima presente en la partitura, especificación en el pipeline de la longitud del tiempo de análisis horizontal, inclusión de la posibilidad de realizar dicho análisis horizontal en ASP y flexibilización de los resultados del módulo de armonización, es decir, en vez de prohibir las soluciones erróneas, se anotarán los errores en la partitura y se escogerá aquella solución que minimice el número de errores. Además se incluirán en este prototipo las correcciones a los errores de ejecución de los módulos no corregidos en el primero.

Se modificó el *parser* substancialmente ya que este imprimía a un fichero según procesaba las notas. Esto no planteaba problema alguno si la subdivisión se especificaba de antemano mediante el parámetro correspondiente, pero sí que resultaba complicado mantener esta aproximación si la unidad de subdivisión debía calcularse al mismo tiempo que se procesaba la partitura en MXML. Se plantearon dos soluciones, o bien incluir en el pipeline en python un análisis

previo a la conversión de MXML a hechos en ASP que dedujese cual era la nota de menor longitud y la usase como parámetro en la llamada al *parser* o bien se modificaba el comportamiento del anterior para realizar simultáneamente ambas tareas.

Se optó por la segunda opción por motivos de coherencia con el sistema, es decir, no incluir funcionalidad innecesaria y replicada en el pipeline, cuya tarea es simplemente manejar las entradas y salidas de los diferentes módulos, y por motivos de eficiencia, ya que como se ha mencionado no hay necesidad de procesar el mismo fichero dos veces, siendo una de ellas en un lenguaje interpretado en vez de compilado, lo que añadiría un sobrecoste temporal evitable.

Los cambios implementados en el *parser* implicaron principalmente incluir un nuevo tipo de dato nota para almacenar la información de las notas de la partitura y una nueva pila que contuviese las notas extraídas del MXML. Una vez procesado todo el fichero de entrada, hallada la nota más breve y almacenadas las notas en la pila, ésta se vacía y se imprimen en el fichero de hechos lógicos de salida teniendo en cuenta la subdivisión pertinente, bien sea esta la calculada o la especificada por parámetro. En caso de especificar una subdivisión no válida, es decir, de mayor longitud que alguna de las notas presentes en la partitura, se imprime por pantalla un mensaje de error y la nota no es subdividida. Esto produce comportamientos no deseados a la hora de realizar la deducción de la armonía, ya que es necesario trabajar siempre con piezas con notas de longitudes iguales a lo largo de toda la partitura.

El módulo de armonización incluye una nueva constante que indica la longitud del intervalo de tiempo mínimo de análisis armónico horizontal, a su vez es posible especificar en la llamada al módulo el valor de esta constante. Se ha modificado, por tanto, la regla que restringe las posibles soluciones para analizar en dichos intervalos de tiempo. Además dicha regla se ha suavizado y en vez de ser una restricción de integridad, esta activa un nuevo predicado error(voz, grado, tiempo) que indica los grados erróneos presentes en la partitura que no encajan con el acorde asignado para la solución. Posteriormente se realiza un proceso de optimización consistente en la minimización de el número de estos predicados de error, es decir, aquellas soluciones con menor número de errores serán las óptimas. Además en caso de no encontrar una solución con cero errores, los errores aparecen también en la salida para que el usuario pueda conocer qué tiempos

contienen notas equivocadas.

Se han incluido en el pipeline opciones tanto para indicar al *parser* una subdivisión específica como para indicar al módulo de armonización el intervalo horizontal de armonización. Se han implementado en el pipeline, con vistas al futuro de módulo de salida, una serie de clases para almacenar los resultados y poder devolverlos más tarde en el formato más conveniente. Error y Chord son clases para almacenar y representar los acordes asignados en la solución así como los errores presentes en la partitura, es decir, aquellas notas que no encajan con los acordes asignados. A su vez, HaspSolution es una clase orientada a almacenar y representar soluciones completas incluyendo una colección de objetos Chord y otra de objetos Error, así como el grado de optimización de dicha solución. Por último, la clase ClaspResult almacena la salida sin procesar de una ejecución del módulo ASP, aunque cuando esta es instanciada, dicha salida se procesa y se crean una colección de objetos HaspSolution conteniendo sólo aquellas soluciones que alcancen un cierto valor de optimización. La relación entre las clases así como sus métodos se detallan en **el diagrama de clase pertinente**.

Se han continuado con las pruebas a los diferentes módulos, dando por válidos los resultados de las pruebas de la anterior iteración se procedió a probar las nuevas funcionalidades de cada módulo.

En el *parser* se comprobó que la autosubdivisión funcionaba correctamente al introducir partituras con diferentes longitudes de notas, además en caso de forzar una subdivisión incorrecta se producía el mensaje de aviso adecuado explicando el problema. Si la subdivisión forzada era correcta funcionaba como debía. El nuevo tipo de datos, nota, así como la pila para almacenar las diferentes notas de la partitura fueron probados junto con esta nueva funcionalidad, ya que fueron implementados expresamente para su funcionamiento.

En el módulo ASP se probaron piezas más y menos complejas, algunas creadas de forma artificial para producir errores y se vio que el nuevo análisis funcionaba y detectaba dichos errores, no obstante seguía produciendo soluciones válidas marcando aquellas notas erróneas.

En cuanto al pipeline se comprobó que las nuevas opciones en la llamada a éste funcionasen correctamente, ya que las dos nuevas opciones son simplemente parámetros que se pasan más adelante al módulo *parser* o al módulo ASP, bastó con realizar una comprobación de validez del valor pasado en la llamada.

Se probaron además las cuatro clases de almacenamiento de las soluciones diseñadas e implementadas para este prototipo, siendo ClaspOutput la única que tuvo que ser probada más exhaustivamente ya que es la que realiza el procesado de la salida del módulo ASP, el resto simplemente cuentan con constructores y funciones de representación textual.

## 3.4. Iteración 4

El tercer prototipo del proyecto, correspondiente a esta iteración pretende no solo refinar la armonización como el anterior sino ampliar funcionalidad de la herramienta. Se ha incorporado un módulo de salida escrito en python al cual el pipeline se encarga de pasarle los datos formateados correctamente para que dicho módulo, haciendo uso del toolkit **music21**, exporte al formato deseado. Se ha optado por este módulo principalmente por la cantidad de opciones de salida que posee, y aunque lo ideal será exportar un fichero MusicXML, la idea de poder generar PDF, MIDI o Lilypond resulta más que atractiva.

El módulo ASP se ha aumentado para incluir generación de notas en un nñumero de voces adicionales que puede ser especificado por parámetro. Además se creó un fichero de conversiones encargado de traducir valores de notas a grados, octavas y viceversa. De este modo los grados generados durante el procesado de la partitura pueden ser traducidos de vuelta a un valor de nota para que el módulo de salida llamado desde el pipeline reconstruya la pieza. Para la generación de notas en las nuevas voces se ha impuesto una única restricción fuerte, que dos notas consecutivas no realicen un salto melódico de dos octavas o más, mientras que mediante predicados de minimización se controla la cantidad de saltos de una quinta realizados por una misma voz. Junto con estas adiciones, se han incluído dos nuevos acordes a la lista de posibles acordes deducidos por el módulo, siendo estos la subdominante séptima (IV7) y la dominante séptima (V7) tanto de los modos mayor como menor.

Se ha diseñado implementado una nueva clase de almacenamiento Note y se ha incluido un método a HaspSolution que transforma una solución al tipo de dato requerido por el *toolkit* music21 para producir los diferentes formatos de salida.

El módulo de salida toma un objeto ClaspResult como entrada y haciendo

uso del método de una de los objetos HaspSolution contenidos en él, representa dicha solución en el formato adecuado.

En el *pipeline* se ha incluido una opción para especificar el número de voces adicionales que deben ser añadidas y otra opción para especificar el formato de salida. Este componente ahora se encarga también de llamar al módulo de salida.

## 3.5. Iteración 5

El cuarto prototipo tiene como objetivo tener en cuenta los tiempos débiles y fuertes de la partitura para refinar ciertas preferencias ya incluidas anteriormente o posibles nuevas preferencias basadas en dichos tiempos. Para esto será necesario modificar el módulo ASP, el parser y el pipeline. Además se refinará el módulo de salida para ajustar la representación en partitura de la solución e incluir mejoras visuales. Como en los anteriores, este prototipo corregirá los errores detectados en el anterior.

En el parser se han incluido dos nuevas funciones principales: análisis de medida de compás y reconocimiento de silencios "verdaderosz silencios completables. La primera atiende a la necesidad de indicarle al módulo ASP la métrica del compás para poder identificar tiempos débiles y fuertes, mientras que la segunda atiende a la carencia de una manera sencilla de representar en música con un símbolo aquellos huecos que siendo silencios en la partitura original deben ser completados por el módulo ASP y no ser tratados como silencios en sí. Para la métrica del compás, se implementó en el *parser* la capacidad de itentificar las diferentes métricas de compases así como el tiempo en el que ocurren, aunque por comodidad y sencillez, se ha asumido que un cambio rítmico en el compás debe ocurrir en todas las voces a la vez. Esto crea predicados "measure(T,N)"que indican para un tiempo T, el número N de figuras que componen el compás. No es necesario especificar la subvidisión ya que esta se fija y normaliza durante el análisis de la partitura. De este modo, aunque el compás fuese un 4/4 (compases de cuatro negras), si la subdivisión detectada y empleada es de corchea, será traducido a 8/8 (compases de ocho corcheas). Para los silencios "verdaderosz los silencios completables se ha optado por una solución relativamente sencilla no solo de identificar por el *parser* sino también fácil de usar por el usuario final de la herramienta, mediante la notación de letras de la pieza, se pueden indicar en

cada voz los intervalos de tiempo en los cuales los silencios deben ser tratados como completables. Para ello solo hace falta escribir los símbolos "[ɾ "].al principio y final del intervalo respectivamente.

Gracias a las mejoras implementadas en el *parser*, el módulo ASP puede diferenciar tiempos débiles de tiempos fuertes, así como los intervalos que debe completar. Se han definido acordemente varios predicados nuevos, como busybeat(V,B) siendo B un tiempo de la voz V que indica aquellos tiempos en los que inicialmente ya hay una nota o silencio, todos aquellos beat(B) que no son busybeat(V,B) son freebeat(V,B), y deben ser completados. Además se han definido los predicados $strong_beat(B)$, $semistrong_beat(B)$ y $weak_beat(B)$ que corresponden a los tiempos fuertes, semifuertes y débiles de cada compás en la partitura. Gracias a esto se han creado predicados que matizan los diferentes errores de la partitura $error_in_strong(V,G,B)$, $error_in_semistrong(V,G,B)$ y $error_in_weak(V,G,B)$ que definen en qué tipo de tiempo ocurren los errores de la pieza y permite minimizarlos con diferentes prioridad (a más fortaleza de tiempo, más prioridad). Por último y para dar más flexibilidad en la búsqueda de la armonización correcta se ha incluído el acorde de dominante séptima (V7) en los modos mayor y menor.

El módulo de salida se ha refinado para representar mejor las notas incluyendo información del tipo de compás, clave y duración de las figuras. Además se colorean en rojo los errores detectados y se anotan en los tiempos adecuados los acordes inferidos por el módulo ASP.

Por último, el pipeline cuenta con una nueva opción timeout que permite especificar un tiempo máximo de búsqueda del óptimo en el módulo ASP, ya que para piezas largas, el espacio de búsqueda crece muchísimo y es necesario poder limitar el tiempo de ejecución. Además, al final de la ejecución, y tras enumerar todas las soluciones escogidas por el módulo ASP, permite al usuario escoger qué solución exportar o mostrar, teniendo por defecto la última, ya que se supone más refinada.

## 3.6. Iteración 6

El quinto prototipo pretende cerrar ya el ciclo de desarrollo del proyecto y pasar únicamente a probar exhaustivamente la herramienta para poder corregir

los fallos de ejecución y realizar mediciones de eficiencia y eficacia. En esta iteración se perfeccionarán algunas funcionalidades de los diferentes módulos y se añadirá un sub-módulo de preferencias melódicas al ya creado módulo ASP.

Para facilitar la entrada de silencios que representan huecos completables por el módulo ASP se ha cambiado el enfoque, y se ha abandonado el delimitado de secciones completables con corchetes en las letras de la canción por suponer algunos problemas al no poder ubicar dichas letras en tiempos en los que haya un silencio, por claridad y por no interferir con las posibles letras de una partitura real no creada ni modificada textitad-hoc para el programa. MusicXML permite marcar elementos de la partitura como no visibles, esto solo afecta a la hora de imprimir en papel dicha partitura y a nivel musical no interfiere con ningún elemento. Además, la visibilidad de una nota o silencio puede ser facilmente alterada en cualquier editor de partituras desmarcando una casilla al clicar sobre dicho elemento, lo cual facilita mucho el marcado de estos tiempos completables. Esto se reflejó en el *parser*, que en vez de contemplar los dos símbolos utilizados anteriormente, ahora solo tiene que comprobar la visibilidad de un elemento para determinar si asignar un tiempo completable a dicho tiempo.

Para refinar la subdivisión en tiempos débiles y fuertes, se reimplementaron tanto en el módulo ASP como en el *parser*, esto fue debido a que no es sencillo establecer dichos tiempos aritméticamente solo teniendo en cuenta la cantidad de notas del compás, sino que también se ha de tener en cuenta el tipo y subdivisión del compás con respecto a la nota de referencia usada para armonizar. Para esto es importante no normalizar el compás leído en el fichero XML y generar nuevos predicados indicando los valores del compás sin modificar. En el módulo ASP se ha incluido, de modo similar a los acordes, una tabla de tipos de compás y su subdivisión teniendo en cuenta el compás y la longitud del tiempo de armonización. Se han incluido en dicha tabla los compases más habituales, pero de nuevo, al igual que con los acordes incluidos en los ficheros correspondientes, pueden ser ampliados si el usuario lo necesitase. Una vez inferido el tipo de subdivisiñon del compás, calcular los tiempos débiles y fuertes se puede realizar de forma aritmética, de modo similar a como se hacía en el prototipo anterior. Debido a la pérdida de información resultante de uniformizar las notas de la partitura no es posible detectar los subtiempos fuertes y débiles dentro de secuencias de corcheas, semicorcheas, etc. Además se ha optado por descartar la inferencia de tiempos

semifuertes ya que solo añadían complejidad a la búsqueda del óptimo y no aportaban mejores resultados, los que en el prototipo anterior eran semifuertes, en este son directamente tiempos fuertes.

Se creó un sub-módulo ASP de preferencias melódicas que busca alcanzar una optimización mayor a la hora de generar nuevas voces o cubrir tiempos completables con algunas mejoras que atienden, principalmente, a la secuencia de notas de una misma voz. Se ha definido el salto melódico de forma genérica y éste es minimizado en notas consecutivas usando el tamaño del salto como peso a la hora de minimizarlo, con esto se logra una melodía más continua y sin saltos erráticos, al mismo tiempo que se limita el espacio de búsqueda considerablemente. Se ha inferido un nuevo predicado tendencia que analiza, dado un par de notas de una voz, si la tendencia melódica es ascendente, descendente o no varía. Además se ha definido otros predicados relacionados que analizan si la tendencia entre dos voces diferentes es la misma o si es contraria, maximizando estos predicados se consigue un ligero efecto de imitación de tendencia entre voces.

Se estudió la posibilidad de incluir también en este sub-módulo de preferencias melódicas la detección de patrones secuenciales armónicos y melódicos comunes como las secuencias de sextas, que siguen un determinado patrón de saltos melódicos en cada voz y que sería interesante completar de forma correcta para buscar una mayor naturalidad en las líneas melódicas de la pieza a la hora de completarla. No obstante se ha descartado inicialmente por incompatibilidad con el análisis armónico, que al subdividir las notas largas a una figura base para poder realizar el análisis en una cantidad arbitraria de tiempos elimina la posibilidad de detectar una secuencia concreta, ya que la secuencia se ve continuamente interrumpida por notas repetidas que no coinciden con el siguiente paso de la secuencia buscada.

Se ha descartado también la detección de apoyaturas con el fin de no contemplarlas como errores por un motivo similar, al uniformizar la longitud de las figuras de la partitura, no es posible detectarlas bien, ya que una de las características de las apoyaturas es que roban"brevemente el tiempo fuerte a una nota representativa del acorde de la armonía. Para detectar dicha brevedad sería necesario conservar la longitud original de las notas, lo que imposibilitaría una armonización correcta.

En el pipeline se ha incluido una nueva opción que permite activar o desactivar este módulo de preferencias melódicas a gusto del usuario.

# Capítulo 4

# Conclusiones

## 4.1. Resultados

## 4.2. Trabajo Futuro