Sistema Compositor de acompañamientos basado en planificación

Proyecto Fin de Carrera Ingeniería Informática
Universidad Carlos III de Madrid

Jesús Manzanares Artolazábal.
NIA - 100033325

Directores:
Sergio Jiménez
Tomás de la Rosa

Tabla de Contenidos

Índice de Tablas

Índice de Ilustraciones

Índice de Códigos.

1 Introducción

Este proyecto se enmarca en el campo de la Inteligencia Artificial. Más concretamente en el de la planificación automática.

La planificación automática, a grandes rasgos, se encarga de buscar una solución (como serie de pasos) a un problema por medio de un ordenador. En este caso, la intención es conseguir que esa planificación obtenga como resultado el acompañamiento de una pieza musical. Para la consecución de este fin habremos de equiparar el ámbito de la música (donde tiene lugar nuestro problema) con el de la planificación automática (donde tienen existencia las herramientas a utilizar).

El matrimonio entre la música y la inteligencia artificial es casi tan antiguo como el nacimiento del segundo. Las aproximaciones que se han llevado a cabo en esta materia tienen muy distintos fundamentos.

El enfoque tomado en este proyecto no es, como en otros muchos casos, un intento de emular la interpretación o estilo de algún músico o tendencia. Más que intentar suplantar al compositor, este proyecto es una curiosa intromisión en las posibilidades de acompañamiento que una pieza puede tener desde el punto de vista de las reglas musicales.

En este aspecto, la composición ha sido siempre una creatividad atribuida a los humanos. No obstante, el acompañamiento, es una creatividad restringida dentro de una serie de reglas de armonía. Gran parte de la música moderna se ha escrito obedeciendo estas reglas. Dado que todo este conocimiento es modelable, vamos a poder representar el conocimiento esa música.

Una persona siempre va a tener un punto de vista parcial sobre cómo acompañar una nota. El punto de vista, además, no se centra en la nota como tal, sino en el entorno que la rodea, es decir, “la canción”. De una manera más escueta: el resto de notas próximas.

En este trabajo vamos a ignorar el conocimiento subjetivo para generar los acompañamientos. Nos olvidaremos de qué suena bien o mal para alguien. Centrándonos únicamente en la satisfacción de las premisas de las reglas armónicas. Así, el problema del acompañamiento, se convierte en un problema aproximable desde una perspectiva científica.

Este proyecto está enfocado de una manera totalmente modular. Esto significa que permite cambiar cada una de las partes y adecuarla a otros requisitos o funcionalidades. Además, la facilidad de definición de nuevas reglas permite ampliar los conocimientos musicales modelados y, eventualmente, conseguir un compositor automático del estilo que se prefiera.

2 Estado del Arte

Esta sección aborda el análisis de algunos de los sistemas de generación de música con ordenador basados en técnicas de Inteligencia Artificial (IA) más representativos. Se distinguen tres subsecciones: La primera está dedicada a los sistemas de composición, la segunda describe los sistemas de improvisación, y la tercera está dedicado a los sistemas de interpretación.

2.1 Sistemas Composicionales

El trabajo de Hiller e Isaacson [Hil] sobre el ordenador ILLIAC (1958), es el más conocido de los pioneros de la música por ordenador. Su principal resultado es la Illiac Suite, un cuarteto de cuerda compuesto siguiendo el sistema de aproximación a la solución de generación y test. El programa generaba notas pseudoaleatoriamente por medio de cadenas de Markov. Las notas generadas eran entonces puestas a prueba por medio de reglas heurísticas de composición clásica, de armonía y contrapunto. Sólo las notas que cumplían las reglas se mantenían. Si ninguna de las notas generadas satisfacía las reglas, se retrocedía en el procedimiento para borrar toda la composición hasta ese punto y se iniciaba de nuevo. Los objetivos de Hiller e Isaacson excluían todo lo relacionado con la expresividad y el contenido emocional. En una entrevista [Sch], Hiller e Isaacson declararon que antes de abordar la cuestión de la expresividad, había que resolver cuestiones más simples en primer lugar.

Posteriormente a este trabajo, muchos otros investigadores basaron sus composiciones en transiciones probabilísticas de Markov, con un éxito bastante limitado, sin embargo, en cuanto a calidad melódica. De hecho, los métodos que confían demasiado en procesos Markovianos no son lo suficientemente completos como para producir música coherente de alta calidad.

No obstante, no todos de los primeros trabajos sobre composición se basaron en criterios probabilísticos. Un buen ejemplo es el trabajo de Moorer [Moo] sobre la generación de melodía tonal. El programa de Moorer generaba melodías simples, junto con progresiones armónicas simples, con repeticiones simples de patrones de notas [Lev]. Este enfoque se basa en la simulación de los procesos de composición humanos utilizando técnicas heurísticas en vez de las cadenas de probabilidad de Markov. Levitt [Lev] también evita el uso de probabilidades en el proceso de composición. Sostuvo que la aleatoriedad tiende a oscurecer en vez de revelar las limitaciones musicales necesarias para representar las estructuras musicales simples. Su trabajo se centra en la descripción de estilos musicales por medio de restricciones.

Hiller desarrolló un lenguaje de descripción que permite expresar transformaciones significativas musicalmente de una entrada, como progresiones de acordes y líneas melódicas, a través de una serie de “relaciones restrictivas” las cuales denomina plantillas de estilo. Aplicó este enfoque para describir una simulación de bajo walking jazz, así como una simulación de ragtime de piano a dos manos. Tanto los primeros sistemas de Hiller e Isaacson como los de Moore se basaron también en enfoques heurísticos.

Ahora bien, posiblemente el más genuino ejemplo de los principios de utilización de técnicas de IA es el trabajo de Rader [Rad]. Rader utilizó programación basada en reglas en su generador de rondas musicales (un canon circular como "Frère Jacques"). La generación de la melodía y la armonía se basaban en reglas que describen cómo notas o acordes se pueden conjuntar.

Los componentes de IA más interesantes de este sistema son la posibilidad de aplicar meta-reglas que determinan la aplicabilidad de las reglas de generación de melodía y acordes, y las reglas de ponderación, las cuales generan un indicador de la similitud de aplicación de una regla por medio

de un peso.

Pioneros en la Inteligencia Artificial como Herbert Simon o Marvin Minsky también publicaron trabajos relevantes para la computación musical. Simon y Sumner [Sim] describieron un patrón de lenguaje musical formal , así como un método de inducción de patrones para descubrir patrones más o menos implícitos en obras musicales. Por ejemplo, un patrón que puede ser descubierto podría ser: "La sección de apertura está en Do Mayor, es continuada por una sección en dominante y, a continuación, una vuelta a la clave original."

Aunque el programa no estaba completo, vale la pena notar que era uno de los primeros en hacer frente a la importante cuestión del modelado de la música, un tema que ha sido y sigue siendo, en gran medida estudiado. Por ejemplo, el uso de modelos basados en gramáticas generativas ha sido y sigue siendo, un importante y útil enfoque en el modelado de la música [Ler].

Marvin Minsky en su ampliamente conocido documento “Música, Mente y Significado” [Min] aborda la importante cuestión de cómo la música impresiona a nuestras mentes. Él aplica sus conceptos de agente y su papel en una sociedad de agentes como una posible aproximación a la cuestión. Por ejemplo, denota que un agente puede simplemente notar que la música tiene un ritmo particular. Otros agentes podrían percibir los pequeños patrones musicales, como repeticiones de un tono y diferencias como una misma secuencia de notas tocadas una quinta más alta.

Su enfoque también tiene en cuenta relaciones más complejas dentro de una pieza musical a través agentes de orden superior capaces de reconocer grandes secciones de la música. Es importante aclarar que en este documento, Minsky no trataba de convencer al lector acerca de la cuestión de la validez de su enfoque; sólo denotaba su posible validez.

Entre los sistemas de composición, hay un gran número que abordan el problema de la armonización automática utilizando varias técnicas de IA. Uno de los primeros trabajos es el de Rothgeb [Rot]. Escribió un programa en SNOBOL para resolver el problema de armonizar el “unfigured bass” (dada una secuencia de notas de bajo, inferir los acordes y la voz principal que acompañan estas notas) por medio de un conjunto de reglas tales como "si el bajo de una tríada desciende un semitono, entonces la siguiente nota de bajo tiene una sexta." El objetivo principal de Rothgeb no era la armonización automática en sí, sino probar la sonoridad computacional de teorías de armonización de dos bajos del siglo XVIII.

Una de las obras más completas sobre armonización es la de Ebcioglu [Ebc]. Desarrollo un método experto, CHORAL, para armonizar corales al estilo de J.S. Bach. Por medio de heurísticas y restricciones CHORAL produce una armonización correspondiente a una melodía de entrada. La implementación del sistema está elaborado utilizando un lenguaje de programación lógica elaborado por el autor del mismo. Un importante aspecto sobre este trabajo es el uso de conjuntos de primitivas lógicas para representar diferentes puntos de visto de la música (acordes, vista melódica, “time-slice view”,...). Estos conjuntos de primitivas lógicas permiten manejar el problema de representar grandes cantidades de conocimientos musicales complejos.

MUSACT [Bha] utiliza redes neuronales para aprender un modelo armónico moderno. Fue diseñado para capturar intuiciones musicales de cualidades armónicas. Por ejemplo, una de las cualidades del acorde de dominante, es crear en el espectador la expectativa de la tónica. Compositores pueden decidir satisfacer o violar estas expectativas en distintos grados. MUSACT es capaz de aprender tales cualidades y de generar una graduación de expectativas en un contexto armónico dado. En HARMONET [Feu], el problema de la armonización es abordado utilizando una combinación de redes neuronales y técnicas de satisfacción de restricciones.

Las redes neuronales aprenden lo que es conocido como funciones armónicas de los acordes (la

función de tónica, dominante ...) y se utilizan restricciones para rellenar las "voces interiores" de los acordes. El trabajo realizado en HARMONET fue extendido en el sistema MELONET [Hor01], [Hor02]. MELONET utiliza una red neuronal para aprender y reproducir estructuras de alto nivel sobre secuencias melódicas. Dada una melodía, el sistema inventa una armonización y variación de cualquier voz coral de estilo barroco. De acuerdo con los autores, HARMONET y MELONET juntas forman un sistema poderoso de composición musical que genera variaciones cuya calidad es similar a las de un organista experimentado.

Pachet y Roy [Pac] también utilizaron métodos de satisfacción de restricciones para armonizar. Estas técnicas explotan el hecho de que el conocimiento sobre, la melodía y la armonización impone restricciones sobre posibles acordes. Sin embargo, la eficiencia es un problema con aproximaciones puramente de satisfacción de restricciones.

En "Using Rules to Support Case-Based Reasoning for Harmonizing Melodies" [Sab] el problema de la aproximación es abordado utilizando una combinación de reglas y CBR. Este enfoque está basado en la observación de que las armonizaciones basadas puramente en reglas normalmente fracasan porque, en general, las reglas no hacen la música, sino que es la música la que hace las reglas. Por tanto, en vez de confiar en un conjunto de reglas imperfectas, ¿por qué no hacer uso de la fuente de las reglas, es decir, las composiciones en sí? CBR permite utilizar ejemplos de composiciones ya armonizadas como casos para nuevas armonizaciones. El sistema armoniza una melodía buscando primero casos similares ya armonizados; cuando no se encuentran casos similares, se buscan reglas de casos generales de armonización. Si ninguna regla se puede utilizar, el sistema falla y retorna al anterior punto de decisión. Los experimentos demostraron que la combinación de reglas y casos tiene un menor número de fracasos en encontrar una armonización apropiada, comparado al uso de cada una de las técnicas por separado. Otra ventaja de la aproximación basada en casos es que cada nueva pieza armonizada de manera correcta puede ser memorizada y estará disponible como un nuevo ejemplo a la hora de armonizar nuevas melodías. En este caso tiene lugar un proceso de aprendizaje por experiencia. De hecho, cuantos más ejemplos tenga el sistema, menos a menudo tendrán que ser consultadas las reglas y, por tanto, menos fallará.

MUSE es también un sistema de aprendizaje que extiende un pequeño conjunto inicial de restricciones sobre la voz principal, aprendiendo un conjunto de "reglas sobre segunda y primera voz". Aprende reordenando la agenda y el particionamiento de reglas para satisfacer un conjunto de restricciones sobre la voz principal. MUSE aprendió satisfactoriamente algunas de las reglas estándar sobre voz principal incluidas en libros de tonalidad musical tradicionales.

Morales-Manzanares et al. [Mor] desarrollaron un sistema llamado SICIB capaz de producir composiciones musicales a partir de movimientos corporales. Este sistema utiliza sensores colocados en el cuerpo del bailarín y reglas de Prolog para casar los gestos con la música. La arquitectura de SICIB también permite interpretaciones en tiempo real.

Ciertamente la composición por ordenador más conocida utilizando IA es la de David Cope [Cop01], [Cop02], el proyecto EMI. Este trabajo se centraba en la emulación de estilos musicales de distintos compositores. Ha compuesto con éxito música de los estilos de Cope, Mozart, Palestrina, Albinoni, Brahms, Debussy, Bach, Rachmaninoff, Chopin, Stravinsky, and Bartok.

Funciona buscando recurrentemente patrones musicales en distintas obras (al menos dos) del mismo compositor. Los patrones descubiertos se llaman firmas. Porque las firmas son dependientes de la localización, EMI utiliza una de las obras del compositor como guía para fijarlos en una localización apropiada en una nueva composición. Para componer los motivos musicales entre las firmas, EMI utiliza un analizador de reglas musicales para descubrir las restricciones utilizadas por el artista en su obra. El analizador cuenta eventos musicales tales como las direcciones de la voz

principal y el uso de patrones de notas repetidos y los representa como modelos estadísticos de las obras analizadas. El programa sigue este modelo de composición para rellenar los espacios que quedan entre las firmas. Para insertarlos apropiadamente, EMI tiene que lidiar con problemas tales como ligar las partes finales e iniciales de las firmas a los motivos que las rodean evitando anomalías estilísticas, manteniendo la moción de las voces, y manteniendo las notas dentro de un rango. La inserción correcta se consigue por medio de una red de transiciones aumentada. Los resultados, aunque no perfectos, resultan bastante consistentes con el estilo del autor.

2.2 Sistemas de Improvisación

Un temprano trabajo sobre improvisación por ordenador es el sistema FLAVORS BAND de Fry [Fry]. FLAVORS BAND es un lenguaje de procedimiento embebido en Lisp, para especificar estilos musicales de jazz y música popular. Su representación de procedimiento permite la generación de partituras en un estilo predefinido haciendo modificaciones sobre una partitura especificada dada como entrada. Permite la combinación de funciones aleatorias y restricciones musicales (acordes, modos, etc...) para generar variaciones improvisadas. El resultado más significativo de FLAVORS BAND fue una interesante interpretación de la linea de bajo, y un solo improvisado, de la composición "Giant Steps" de Miles Davis.

GENJAM [Bil] construye un modelo de un músico de jazz aprendiendo a improvisar por medio de algoritmos genéticos. Un oyente humano juega el papel de una función de fitness dando una puntuación a las improvisaciones que es el "offspring". Papadopoulos y Wiggins [Pap] también usaron algoritmos genéticos para improvisar melodías de jazz sobre una progresión de acordes dada. Contrariamente a GENJAM, el programa incluye una función de fitness que automáticamente evalúa la calidad de las improvisaciones con ocho aspectos diferentes; el contorno melódico, la duración de la nota, los intervalos, etc …

Franklin [Fra] utiliza redes neuronales recurrentes para aprender como improvisar solos a partir de transcripciones de las improvisaciones de Sonny Rollins. Un algoritmo de refuerzo al aprendizaje es utilizado para refinar el comportamiento de las redes neuronales. El sistema puntúa las improvisaciones según criterios de armonía de jazz y del estilo de Rollins. La falta de interactividad con un improvisador humano, de los enfoques anteriores ha sido criticado [Tho] en base a que, estos métodos eliminan al músico del plano físico y espontáneo de la creación de la melodía. Aunque es verdad que las características fundamentales de la improvisación es la espontaneidad en tiempo real de la creación de la melodía, también es verdad que la interactividad no estaba enfocada hacia ese punto de vista. De todas formas, podían generar improvisaciones muy interesantes.

Thom [Tho] con su sistema BAND-OUT-OF-A-BOX (BOB) afronta el problema de la interactividad en tiempo real entre BOB y un músico humano. En otras palabras, BOB es un sistema de acompañamiento para improvisaciones en tiempo real. El enfoque de Thom está basado en la teoría psicológica sobre la improvisación de jazz de Johnson-Laird [Joh02]. Esta teoría se opone a la opinión de que la improvisación se compone de la reordenación y transformación de licks aprendidos bajo las restricciones de una armonía. En cambio propone un modelo estocástico basado en una búsqueda avariciosa sobre un espacio restringido de posibles notas en un punto dado del tiempo. La mayor aportación de Thom es que su sistema aprende estas restricciones, y por tanto el modelo estocástico, de un músico humano por medio de un algoritmo de clustering. El modelo aprendido es utilizado para abstraer los solos en modos de interpretación especificados por el usuario. Los parámetros de este modelo aprendido son entonces añadidos a un proceso estocástico que genera los solos respondiendo a solos tocados por un humano de una duración de cuatro compases. BOB ha sido exitosamente evaluado en intercambios en tiempo real en dos estilos

distintos; el del saxofonista Charlie Parker y el violinista Stephane Grapelli.

Otro sistema de improvisación interactiva fue desarrollado por Dannenberg [Dan01]. La diferencia con el enfoque de Thom es que en el sistema de Dannenberg, la generación musical está principalmente dirigida por los objetivos del compositor en vez de las metas del ejecutor. El sistema de improvisación interactivo de Wessel [Wes] es más cercano al de Thom puesto que enfatiza el acompañamiento y la improvisación en vivo.

2.3 Sistemas Interpretativos

Los sistemas combinatorios y de improvisación descritos hasta el momento, aunque generaban una salida audible en muchos casos (primeramente por medios electrónicos y más tarde con instrumentos MIDI), no abordaban específicamente el tema de la expresividad. Sin embargo, para los sistemas interpretativos, la expresividad es una preocupación obligatoria porque el cerebro está principalmente interesado en los cambios. Las neuronas auditivas, como otras neuronas del cerebro, se estimulan constantemente incluso en entornos silenciosos. Lo que verdaderamente importa no es la velocidad con que son estimuladas esas neuronas pero los cambios en esta estimulación. Hay neuronas cuyo régimen de estímulos se modifica únicamente cuando cambia la frecuencia del sonido o la intensidad aumenta o disminuye. Otras neuronas reaccionan de manera similar únicamente cuando un sonido se repite. Por el contrario, la mayoría de las neuronas auditivas primarias también exhiben lo que se conoce como habituación [Baa]. Cuando las neuronas reciben seguidamente el mismo estímulo, su tasa de estimulación, en lugar de permanecer constante, disminuye con el tiempo, lo que significa que ignoran un sonido a menos que se manifiesta algún tipo de novedad o renovación en sus características.

Por lo tanto, no es de extrañar que la música se vuelva más interesante cuando no es demasiado repetitiva, es decir, cuando contiene alteraciones en la dinámica, tono y ritmo. Este conjunto de cambios podría explicar en parte la razón por lo que la música sintetizada es mucho menos interesante que la música interpretada por el hombre: Un verdadero instrumento envía a la corteza auditiva muchos más estímulos ante los que reaccionar que la música sintetizada. De hecho, los llamados recursos expresivos proporcionan una rica fuente de cambios para nuestros cerebros.

Vale la pena denotar que ha habido muchos menos trabajo en técnicas de IA para la interpretación que en la IA para la composición y la improvisación y que todos los trabajos son recientes. Uno de los primeros intentos para proporcionar un alto nivel de transformaciones musicales utilizando un sistema basado en reglas es el de Johnson [Joh01]. Ella desarrolló un sistema experto para determinar el “tempo” y la articulación que deben aplicarse al interpretar las fugas de Bach "The Well-Tempered Clavier". Las reglas se obtuvieron a partir de dos músicos expertos. La salida da el tempo base y una lista de instrucciones sobre la duración de las notas y la articulación que deberían ser obedecidas por un intérprete humano.

Los resultados coinciden, en gran medida, con los comentario dados sobre "The Well-Tempered Clavier" en ediciones conocidas de éste. La principal limitación de este sistema es su falta de generalidad, ya que sólo funciona bien para fugas escritas en cuatro por cuatro. Para las diferentes medidas, las reglas deberían ser diferentes. Otra consecuencia evidente de esta falta de generalidad es que las reglas son aplicables solamente a las fugas de Bach.

La labor del grupo KTH de Estocolmo ([Bre01]; [Fri01]; [Fri02]; [Fri03]) es uno de los más conocidos esfuerzos a largo plazo en sistemas de interpretación. Su actual proyecto DIRECTOR MUSICES incorpora reglas para el tempo, dinámica, articulación y transformaciones restringidas de MIDI. Estas reglas se infieren, tanto de conocimientos teóricos de música experimental y de

formación, en especial mediante el enfoque llamado análisis por síntesis. Las reglas se dividen en tres clases principales: (1) reglas de diferenciación, las cuales aumentan las diferencias entre tonos de escala; (2) reglas de agrupación, que muestran qué tonos van juntos, y (3) "reglas de ensemble", que sincronizan las diferentes voces en conjunto. Otro esfuerzo a largo plazo es el proyecto dirigido por G. Widmer [Wid01], [Wid02]. Es otro ejemplo de la utilización de reglas para ejecutar transformaciones de tempo y dinámica. El enfoque adoptado en este proyecto fue primero grabar y preprocesar una gran cantidad de actuaciones musicales (con más de 400000 notas) de alta calidad. Ese gran número de ejemplos es utilizado para inducir reglas de interpretación por medio de técnicas de aprendizaje automático. El proyecto ya ha obtenido resultados muy prometedores.

Canazza et al. [Can] desarrolló un sistema para analizar la forma en que las intenciones expresivas del músico se reflejan en la interpretación. El análisis revela dos diferentes dimensiones expresivas: (1) uno relacionado con la energía (dinámica) y (2) y el otro relacionado con la cinética (rubato) de la pieza. Los autores también desarrollaron un programa para generar actuaciones expresivas en función de estas dos dimensiones.

La labor de Dannenberg y Derenyi [Dan02] es también un buen ejemplo de articulación de transformaciones construidas manualmente utilizando reglas. Ellos desarrollaron una trompeta sintetizadora que combina un modelo físico con un modelo de interpretación. El objetivo del modelo es generar información de control para el modelo físico a través de una colección de reglas manualmente extraídas del análisis de una colección de grabaciones de interpretaciones humanas. Otro enfoque adoptado para realizar el tempo y la dinámica de transformación es el uso de técnicas de redes neuronales. En Bresin [Bre02], un sistema simbólico que combina reglas de decisión con redes neuronales se aplica para simular el estilo de los artistas intérpretes o ejecutantes de piano real. La salida de las redes neuronales expresa desviaciones en el tiempo y el volumen. Las neuronas de entrada de estas redes son retroalimentadas con las salidas de la red.

Podemos ver que, excepto para los trabajos del grupo KTH, que considera tres recursos expresivos, los otros sistemas se limitan a dos recursos como el rubato y la dinámica o rubato y articulación. Esta limitación tiene que ver con el uso de las reglas. De hecho, el principal problema con las aproximaciones basadas en reglas es que es difícil encontrar reglas suficientemente generales para capturar la variedad presente en las distintas interpretaciones de una misma pieza por el mismo músico, e incluso las diferencias dentro de una misma interpretación [Ken].

Por otra parte, los diferentes recursos expresivos interaccionan unos con otros. Es decir, las reglas de dinámica cambian por sí solas cuando el rubato se tiene en cuenta. Obviamente, a causa de esta interdependencia, cuantos más recursos expresivos se intentan modelar, más difícil resulta encontrar las reglas adecuadas.

López de Mántaras y Lluis Arcos [Lop], con sus sistema SAXEX, en vez de intentar hacer el conocimiento de esos recursos como un sistema de reglas, toman como fuente grabaciones monofónicas de actuaciones humanas. Con ese nuevo enfoque pretendían hacer frente a los cinco recursos expresivos más importantes: (1) dinámica, (2) rubato, (3) vibrato, (4) la articulación, y (5) ataque de las notas.

3 Objetivos

Como ya se ha comentado en la introducción la intención principal de el proyecto es acompañar una pieza musical de manera automática empleando técnicas de planificación. Tenemos, para llegar a este fin, que definir dos objetivos principales:

- Representar una melodía musical utilizando el lenguaje estándar de planificación automática.
- Representar un conjunto de reglas armónicas como acciones de planificación que nos permitan obtener un acompañamiento.

Hay que tener en cuenta que surgen una serie de objetivos secundarios que también debemos cubrir para poder completar los primeros:

- Traducir de un fichero MIDI al lenguaje estándar de planificación automática PDDL.
- Desarrollar un GUI que nos facilite la generación de reglas de acompañamiento.
- Obtener una salida MIDI a partir de los planes generados con nuestro dominio de planificación

Los primeros objetivos son bastante claros. Dado que queremos acompañar una pieza musical por medio de planificación, debemos en primer lugar poder expresar la pieza musical como una entrada para un planificador.

Las entradas posibles que puede tener un planificador son dos. La definición del dominio y la definición del problema. La definición del dominio, a parte de los elementos de representación musical contendrá las reglas de acompañamiento. La definición del problema contendrá la pieza que queremos acompañar.

Los objetivos secundarios surgen de las dificultades e impedimentos del desarrollo de lo anterior. En primer lugar la traducción de MIDI al dominio nuevo nos facilita la obtención de melodías. La codificación de cada una de las piezas conlleva un esfuerzo considerable. Dado que la mayoría de las propiedades de las notas que utiliza el dominio tienen su origen en la especificación MIDI, el esfuerzo de codificar un traductor es semejante al de hacer las piezas a mano. Además nos asegura un cierto grado de seguridad (importante a la hora de depurar).

La generación de las reglas de acompañamiento de manera dinámica es un concepto similar al anterior. El proyecto define una base de datos de tal manera que los acordes se pudiesen guardar y clasificar y seleccionar para acompañar una pieza. Tiene una intención para líneas futuras principalmente. El concepto de generar de manera dinámica significa que en cada prueba podemos seleccionar qué reglas tomarán partido en la planificación en forma de acciones.

La creación de una interfaz gráfica de usuario para la manipulación de las partes anteriores nos ofrece la oportunidad de tener cada una de las partes integradas en un todo y nos facilita el uso.

Por último la obtención de una salida MIDI a partir del plan que se ha creado es obligatorio para poder juzgar la “musicalidad” de los acompañamientos.

La “musicalidad” es la forma en la que evaluaremos la calidad de las salidas. Es un concepto subjetivo, en términos objetivos consideraremos una salida acompañamiento correcta siempre que cumpla las reglas armónicas establecidas.

4 Desarrollo

El punto de desarrollo cubre los contenidos teóricos sobre los que se fundamenta este proyecto además de una introducción al abordaje de los objetivos.

4.1 Contenidos

En esta sección se realiza una introducción a los contenidos teóricos en los que se sustenta este proyecto. En primer lugar realizo una introducción a MIDI y sus características, y la importancia que tiene para cualquier desarrollo musical desde un punto de vista electrónico. En segundo lugar se abordan los conceptos de planificación automática relevantes para el proyecto. En la subsección siguiente describo el lenguaje Java y las librerías de las que se ha nutrido el desarrollo de las salidas. Por último explico las bases de la teoría de la música sobre las que se ha construido el sistema de acompañamientos.

4.1.1 MIDI

MIDI (Musical Instrument Digital Interface) es un formato electrónico desarrollado en la época de los 70 para expresar en términos digitales eventos musicales. Desde su desarrollo hasta la fecha, no ha sufrido modificaciones mayores. Cualquier proyecto musical abordado en la actualidad en un plano informático, está sin duda, directa o indirectamente influenciado por su especificación. De igual manera, compatibilizar con el formato es garantizar una integración con el gran abanico de aplicaciones musicales que existe en la actualidad.

En esta sección daré una perspectiva histórica del formato y además una introducción a los aspectos técnicos del mismo.

4.1.1.1 Historia

Hacia finales de la década de los 70 los instrumentos de música electrónica comenzaron a estar al alcance del público con precios asequibles y se volvieron muy comunes. Sin embargo, los dispositivos de unos fabricantes no eran compatibles con los del resto de marcas y no se podían interconectar. Algunos modelos incorporaron interfaces que consistían en controladores de voltaje que seguían varios estándares (1 voltio por octava, un hertzio por voltio, ...); relojes analógicos, disparadores y puertas de señal. También interfaces propietarios como el DCB (ditigal control bus) de Roland, Oberheim y Yamaha también incorporaban otros diseños.

En 1981, Dave Smith, un ingeniero de audio y diseñador de sintetizadores que trabajaba para Sequential Circuits, Inc, propuso un estándar digital para instrumentos musicales en un articulo para la revista Audio Engineering Society. La especificación MIDI 1.0 fue publicada en 1983.

Desde entonces, la tecnología MIDI ha sido estandarizada y es mantenida por la MMA (MIDI Manufacturers Association). Todos los estándares MIDI son desarrollados y publicados por la MMA en los Estados Unidos, el MIDI Committee de la asociación de industrias de la música electrónica (AMEI) en Japón. La principal referencia de MIDI es la versión 96.1 de The Complete MIDI 1.0 Detailed Specification.

A principios de los 80, el estándar MIDI fue la principal razón para que desapareciesen las paredes de sintetizadores tras las cuales se encontraban los teclistas. Gracias al advenimiento del MIDI, con el cual los instrumentos se podían controlar desde un solo teclado, las marcas empezaron a producir

los sintetizadores en formato rack.

MIDI también ha facilitado el desarrollo de secuenciadores hardware y de ordenador. Estos pueden ser usados para grabar, editar y reproducir actuaciones. En los años inmediatamente posteriores a la publicación del estándar, aparecieron interfaces MIDI para las plataformas Apple Macintosh, Commodore 64 y PC-DOS, que permitían el desarrollo de un mercado más poderoso, barato y extendido basado en secuenciadores de ordenador. El Atari ST vino incorporado con puertos MIDI de serie y era muy común en estudios de grabación caseros. El MIDI Timecode hacia posible la sincronización de dispositivos MIDI. Era esta una implementación del SMTPE utilizando mensajes MIDI. Desde entonces ha sido el estándar para la sincronización de música digital. [Mid01]

4.1.1.2 Formato MIDI

Todos los dispositivos compatibles con MIDI, instrumentos musicales, controladores y software siguen la misma especificación MIDI: la versión 1.0 [Son]. Es por ello que interpretan cualquier mensaje MIDI que reciban de igual manera y que por tanto se pueden comunicar y entender los unos a los otros. Por ejemplo, si una nota es tocada en un controlador MIDI, sonará en el tono correcto en los instrumentos cuya entrada MIDI este recibiendo esa información.

Cuando se toca algo en un controlador MIDI se envía un mensaje de canal MIDI a través de su salida a la entrada del instrumento. Una secuencia común es por ejemplo la pulsación y liberación de una nota en un teclado.

Otros parámetros pueden ser transmitidos. Por ejemplo el bending de una nota o su cambio de volumen. Cuando esto ocurre se envía también un mensaje de canal MIDI con la información, el instrumento entonces hace los cambios sobre el sonido pertinentes. El músico por tanto no tiene que preocuparse de los detalles de esta gestualización del sonido, simplemente tiene que utilizar los elementos del controlador MIDI.

Toda conexión MIDI es un camino unidireccional desde el conector de salida (MIDI OUT) del emisor hasta el de entrada del receptor (MIDI IN). Cada una de estas conexión puede transmitir un flujo de mensajes MIDI, donde la mayoría representan un evento musical o gesto como nota-on, nota-off, cambio de controlador de valor (entre ellos el volumen, pedal, la modulación de señales, etc...), pitch bend , cambios en el programa, aftertouch, canal de presión. Todos estos mensajes incluyen número de canal. Hay 16 canales posibles en el protocolo. Los canales son utilizados para separar "voces" o "instrumentos", algo así como pistas en una mesa de mezclas.

La capacidad de multiplexar 16 "canales" en un solo cable hace posible el control de varios instrumentos al mismo tiempo utilizando una única conexión MIDI. Cuando un instrumento MIDI es capaz de producir varios sonidos independientes o "voces" al mismo tiempo (un instrumento multitimbral), los canales MIDI se utilizan para hacer frente a estas secciones de forma independiente. (Esto no debe confundirse con "polifónicos", la capacidad de reproducir varias notas simultáneamente en la misma "voz".)

4.1.1.2.1 Formato de los datos

Todos los datos se guardan en formato Big-Endian, con el MSB (“Most Significant Byte”) a la izquierda. Muchos de los valores, tienen una longitud variable. Estos datos utilizan los bits menos significativos para guardar datos y el más significativo para informar de la finalización o continuación de información.

La siguiente tabla muestra un ejemplo de esta situación:

Valor		Longitud variable	
Hex	Bin	Hex	Bin
00	000000	00	000000
C8	11001000	8148	10000001 01001000
100000	00010000 00000000 00000000	C080 00	11000000 10000000 00000000

Tabla 1. Valores de ejemplo y sus equivalentes en longitud variable.

Un dato de longitud variable puede, como mucho, ocupar 4 bytes. Esto significa que el valor más alto que podemos alcanzar es 0x0FFFFFFF, que tiene la representación 0xFF 0xFF 0xFF 0x7F

4.1.1.2.2 Estructura de los ficheros

Los ficheros MIDI están divididos en paquetes (“chunks”). Cada paquete está precedido por una cabecera de 8 bytes. Contiene un ID de 4bytes que identifica el paquete. A continuación otros 4 bytes indican la longitud en bytes del paquete (sin incluir la cabecera).

4.1.1.2.2.1 Cabecera del fichero

El paquete de cabecera del fichero contiene información sobre la canción completa. Incluye el tipo de formato MIDI, número de pistas y división del tiempo. Sólo hay una cabecera de fichero (a distinguir de las cabeceras de sección) y se encuentra al inicio del fichero. La siguiente tabla muestra un resumen de la organización de la cabecera, después se encuentran sus partes explicadas en profundidad.

Offset	Long.	Tipo	Descripción	Valor
0x00	4	char[4]	Id de paquete	"MThd" (0x4D546864)
0x04	4	dword	Tamaño del paquete	6 (0x00000006)
0x08	2	word	Tipo de formato	0 - 2
0x10	2	word	Número de pistas	1 - 65535
0x12	2	word	Divisiones del tiempo	Ver explicación

Tabla 2. Formato de paquete de cabecera MIDI.

Id de paquete y tamaño

El id de paquete de la cabecera es siempre “Mthd” (0x4D546864) y su tamaño es siempre 6 porque siempre contiene las mismas 3 variables (tipo de formato, número de pistas y división del tiempo)

Tipo de Formato

El formato MIDI utilizado puede ser 0, 1 o 2. Describe cómo se debe interpretar la siguiente información.

En el tipo 0, el fichero tiene una sola pista con todos los eventos para la canción. Incluye el título de la canción, compás, tempo y eventos musicales.

En el tipo 1, el fichero debería tener dos o más pistas. La primera (por convención) contiene información sobre la canción; título, compás, tempo, etc... La segunda y siguientes contienen un título, eventos musicales, etc..., específicos de cada una de las pistas.

El tipo 2 es una combinación de los dos anteriores. Contiene múltiples pistas, pero cada pista representa una sección diferente que no tiene por que reproducirse de manera simultánea al resto. Este formato está orientado para guardar patrones de batería y otras secuencias multipatrón.

Número de pistas

El número de pistas que contiene el fichero se guarda a continuación. En un fichero de tipo 0, el valor será siempre 1. Los fichero de tipo 1 y 2 podrán contener hasta 65,536 pistas (0xFFFF).

Divisiones del tiempo

La tercera y última palabra de la cabecera es ligeramente más complicada que las anteriores. Contiene la división del tiempo utilizada para transformar las marcas de tiempo de los eventos en su medida real. El valor dado representa bien ticks por beat o frames por segundo. Si el bit más significativo del valor es 0, los siguientes 15 bits describen el tiempo en ticks por beat. Si no (1) en frames por segundo.

La razón por la que existen dos formatos de definición del tiempo es el entorno en el cual se utilizará la música. Ticks per beat es puramente musical. Frames por segundo es una medida cinematográfica y es útil utilizarla cuando se está sincronizando música con video.

Ticks per beat se traducen en el numero de ticks de reloj (o posiciones delta de la pista – sección de pista) por cada cuarto de nota musical. Normalmente su valor oscila entre 48 y 960. Este valor puede ser mucho más alto en caso de que se combine MIDI y audio para trabajar más fácilmente con la combinación.

Frames por segundo se define descomponiendo los 15 bytes resultantes en dos. Los primeros 7 bits definen el valor para el número de SMPTE frames; puede ser 24, 25, 29 o 30. El bit sobrante define cuántos ticks hay por cada frame.

Así, una división del tiempo marcada como 0x9978 define, en primer lugar que la medida está en SMPTE frames por segundo, los siguientes 7 bits nos indican que hay 25 frames, y el siguiente byte 120 ticks por frame.

4.1.1.2.2.2 Paquete de pista

Los paquetes de pista contienen toda la información de cada pista individual, como ya se dijo antes. El siguiente resumen muestra la organización de una pista:

Offset	Long.	Tipo	Descripción	Valor
0x00	4	char[4]	Id del paquete	"MTrk" (0x4D54726B)
0x04	4	Dword	Tamaño del paquete	Ver explicación
0x08	Datos sobre los eventos (ver explicación)			

Tabla 3. Formato de Paquete de pista MIDI.

Id de sección y tamaño

El id de paquete es siempre “Mtrk” (0x4D54726B) y el tamaño varía dependiendo del número de bytes utilizados por todos los eventos que contiene el track.

Información de evento de pista

Los eventos de la pista son una sucesión de eventos MIDI. Definen una secuencia de información sobre qué y cómo se reproduce.

4.1.1.2.2.3 Eventos MIDI

Los eventos de una pista describen todo el contenido musical de un fichero MIDI. Cambios en el tiempo, títulos de pista, eventos musicales. Cada uno contiene una marca de tiempo (delta time), un tipo de evento y dependiendo del tipo de evento pueden contener más información específica a tal.

4.1.1.2.2.3.1 Delta-Times

Esta representado por un dato de longitud variable. Determina cuando un evento debe ser reproducido relativamente al último evento de la pista. Un delta time de 0 significa que se debe reproducir simultáneamente al último evento. El delta time del primer evento de una pista determina cuánto tiempo esperar hasta reproducir el primer evento.

Los eventos que dependen del tiempo (títulos de pista, información de copyright...) también van acompañados de la marca de tiempo, solo que ésta está puesta a 0. Estos eventos deben ir los primeros en la secuencia de eventos.

Es importante recordar que los delta times son tiempos relativos. Dependen directamente de varios factores: la división del tiempo descrita en la cabecera del fichero y el tempo de la pista (definido en un evento MIDI). Si no existe tal evento MIDI que defina el tempo, se utiliza 120 beats por minuto por defecto.

4.1.1.2.2.3.2 Tipos de Evento

Existe tres tipos de eventos: eventos de Canal, eventos Exclusivos del Sistema y Meta eventos.

Eventos de Canal

Las informaciones musicales tales como reproducir una nota o ajustar la modulación de un canal y los valores asociados se definen en un evento de canal MIDI. Cada evento de canal consiste en un delta time (como todos) y dos o tres bytes que determinan el canal MIDI al que corresponde, el tipo de evento que es y valores específicos. La siguiente tabla muestra el formato:

Delta Time	Tipo de dato	Canal MIDI	Parámetro 1	Parámetro 2
Longitud variable	bits	4 bits	1 byte	1 byte

Tabla 4. Formato de evento de canal MIDI.

Los eventos de Canal son los tipos de datos más comunes. La siguiente lista incluye un resumen de los siete eventos que existen y sus posibles parámetros.

Tipo de evento	Valor	Parámetro 1	Parámetro 2
Note Off	0x8	note number	velocity
Note On	0x9	Note number	velocity
Note Aftertouch	0xA	note number	aftertouch value
Controller	0xB	Controller number	Controller value
Program Change	0xC	program number	not used
Channel Aftertouch	0xD	Aftertouch value	Not used
Pitch Bend	0xE	pitch value (LSB)	pitch value (MSB)

Tabla 5. Eventos de canal MIDI.

Aunque todos los eventos de canal siguen el mismo patrón basico, cada uno requiere una explicación más detallada. A continuación se describe cada uno detalladamente.

Note Off

El evento Note Off se usa para señalar cuando una teclas se deja de pulsar. Este tipo de eventos tienen dos parámetros exactos a los eventos Note On. El número de nota especifica cuál de las 128 teclas se está pulsando y la velocidad determina cuan fuerte se soltó. El número de nota señala una nota que se estaba reproduciendo y la velocidad se ignora; aunque, para algunos instrumentos, es necesaria para calcular su fase de 'release'.

Note Off	MIDI Channel	Note Number	Velocity
8 (0x8)	0-15	0-127	0-127

Tabla 6. Rango de valores del evento Note Off.

Note On

El evento Note On se utiliza para señalar la pulsación de una tecla. Tiene dos parámetros, como se dijo antes, el número de nota y la velocidad. La velocidad determina cuán rápido/fuerte se presionó la tecla y así se convierte en el volumen e intensidad de la nota. El número de nota se transforma en la altura.

Note On	MIDI Channel	Note Number	Velocity
9 (0x9)	0-15	0-127	0-127

Tabla 7. Rango de valores del evento Note On.

Note Aftertouch

El evento Aftertouch se usa para indicar un cambio en la presión de una tecla. Tiene dos parámetros. El número de la nota (para saber a qué nota afecta) y la presión nueva con la que se aplica; varía de 0 a 127. Normalmente ayuda a la expresividad de las notas, por ejemplo añadiendo modulación a la fase de 'sustain' de la nota.

Note Aftertouch	MIDI Channel	Note Number	Amount
10 (0xA)	0-15	0-127	0-127

Tabla 8. Rango de valores del evento Aftertouch.

Controller (Controlador)

El evento de controlador cambia el estado de un canal MIDI. Hay 128 controladores los cuales definen diferentes propiedades del canal: volumen, pan, modulación, efectos, etc... Este tipo de eventos tiene dos parámetros. El número de controlador especifica que control está cambiando y el valor su nueva posición.

Controller	MIDI Channel	Controller Type	Value
11 (0xB)	0-15	0-127	0-127

Tabla 9. Rango de valores del evento Controlador.

La siguiente lista muestra todos los tipos de controladores:

Valor	Tipo de controlador
0 (0x00)	Bank Select
1 (0x01)	Modulation
2 (0x02)	Breath Controller
4 (0x04)	Foot Controller
5 (0x05)	Portamento Time
6 (0x06)	Data Entry (MSB)
7 (0x07)	Main Volume
8 (0x08)	Balance
10 (0x0A)	Pan
11 (0x0B)	Expression Controller
12 (0x0C)	Effect Control 1
13 (0x0D)	Effect Control 2
16-19 (0x10-0x13)	General-Purpose Controllers 1-4
32-63 (0x20-0x3F)	LSB for controllers 0-31
64 (0x40)	Damper pedal (sustain)
65 (0x41)	Portamento
66 (0x42)	Sostenuto
67 (0x43)	Soft Pedal
68 (0x44)	Legato Footswitch
69 (0x45)	Hold 2
70 (0x46)	Sound Controller 1 (default: Timber Variation)
71 (0x47)	Sound Controller 2 (default: Timber/Harmonic Content)
72 (0x48)	Sound Controller 3 (default: Release Time)
73 (0x49)	Sound Controller 4 (default: Attack Time)
74-79 (0x4A-0x4F)	Sound Controller 6-10
80-83 (0x50-0x53)	General-Purpose Controllers 5-8
84 (0x54)	Portamento Control
91 (0x5B)	Effects 1 Depth (formerly External Effects Depth)
92 (0x5C)	Effects 2 Depth (formerly Tremolo Depth)
93 (0x5D)	Effects 3 Depth (formerly Chorus Depth)
94 (0x5E)	Effects 4 Depth (formerly Celeste Detune)
95 (0x5F)	Effects 5 Depth (formerly Phaser Depth)
96 (0x60)	Data Increment
97 (0x61)	Data Decrement
98 (0x62)	Non-Registered Parameter Number (LSB)

99 (0x63)	Non-Registered Parameter Number (MSB)
100 (0x64)	Registered Parameter Number (LSB)
101 (0x65)	Registered Parameter Number (MSB)
121-127 (0x79-0x7F)	Mode Messages

Tabla 10. Tipos de Controlador MIDI definidos.

Program Change

El evento de cambio de programa se usa para cambiar el programa (instrumento/parche). Sólo tiene un parámetro que especifica el nuevo valor de instrumento/parche.

Program Change	MIDI Channel	Program Number
12 (0xC)	0-15	0-127

Tabla 11. Rango de valores del evento Program Change.

Channel Aftertouch

El evento de aftertouch de canal es similar al de nota pero a diferencia de que afecta a todas las notas de un canal. Este tipo de eventos sólo tiene como parámetro el valor de aftertouch.

Channel Aftertouch	MIDI Channel	Amount
13 (0xD)	0-15	0-127

Tabla 12. Rango de valores del evento Aftertouch de Canal.

Pitch Bend

El evento 'pitch bend' es similar a un evento de controlador. Tiene dos bytes para describir su valor. El valor pitch se define por los dos valores del evento de canal concatenando los 7 bits menos significativos del segundo parámetro y los del segundo, quedando primero los del segundo. Esta concatenación permite un aumento de la exactitud, valores entre 0 y 16383. La modificación en la altura afecta a todas las notas del canal MIDI. La altura de la nota puede incrementar o decrementar dependiendo de si el valor está por encima o por debajo de 8192. Normalmente la modificación se encuentra en el rango de 2 semitonos hacía arriba o abajo.

Pitch Bend	MIDI Channel	Value (LSB)	Value (MSB)
14 (0xE)	0-15	0-127	0-127

Tabla 13. Rango de valores del evento Pitch Bend.

Meta Events

Los meta eventos son eventos que en un principio no deberían ser enviados por una conexión MIDI. Estos eventos se identifican por 0xFF y tienen tamaño un tamaño de parámetros variable el cuál se define después del tipo de evento. Actualmente están definidos 15 tipos de meta eventos.

Meta Evento	Tipo	Longitud	Datos
255 (0xFF)	0-255	Longitud variable	Específico del mensaje

Tabla 14. Valores de Meta Evento.

Sequence Number (Número de secuencia)

Estos meta eventos definen el número de patrón de un fichero MIDI de tipo 2 o el número de secuencia cuando se trata de un fichero MIDI de tipo 1 o 0. Siempre debe tener un delta time de 0 y venir antes de todos los eventos de canal y de los eventos con delta time no nulo.

Meta Evento	Tipo	Longitud	Datos	Meta Evento
255 (0xFF)	0 (0x00)	2	0-255	0-255

Tabla 15. Valores del Meta Evento Número de Secuencia.

Text Event (Evento de texto)

Los eventos de texto definen texto que por cualquier razón vaya incluido en las pistas, notas, comentarios, etc... La cadena de texto suele ser en formato ASCII pero sus caracteres deben tener sólo dos bytes.

Meta Evento	Tipo	Longitud	Datos
255 (0xFF)	1 (0x01)	Longitud de la cadena	Texto ASCII

Tabla 16. Valores del Meta Evento Texto.

Copyright Notice (Nota de Copyright)

Este evento define la información de copyright, incluyendo el símbolo © (0xA9), el año y el autor. Siempre debería estar en el paquete de la primera pista, tener un delta time de 0 y estar antes de todos los eventos de canal y los eventos con delta time no nulo.

Meta Evento	Tipo	Longitud	Datos
255 (0xFF)	(0x02)	Longitud de la cadena	Texto ASCII

Tabla 17. Valores del Meta Evento Copyright Notice.

Sequence/Track Name (Nombre de pista/secuencia)

Este meta evento define el nombre de la secuencia en MIDI de tipo 0 y 2 o de la pista en MIDI de tipo 1. Define el nombre de la pista cuando aparece en alguna pista después de de la primera en un fichero de tipo 1. Siempre debería aparecer con un delta time de 0 y antes de todos los eventos de canal y los eventos con delta time no nulo.

Meta Evento	Tipo	Longitud	Datos
255 (0xFF)	3 (0x03)	Longitud de la cadena	Texto ASCII

Tabla 18. Valores del Meta Evento Nombre de Secuencia/Pista.

Instrument Name (Nombre de Instrumento)

Este meta evento define el nombre del instrumento usado en la pista actual. Puede usarse en conjunción al evento de prefijo de canal para definir qué instrumento está siendo usado en un canal.

Meta Evento	Tipo	Longitud	Datos
255 (0xFF)	(0x04)	Longitud de la cadena	Texto ASCII

Tabla 19. Valores del Meta Evento Nombre de Instrumento.

Lyrics (Letra)

Este meta evento define las letras de una canción y normalmente define una silaba a grupo de ellas por cuarto de nota. Este evento se puede utilizar como equivalente a una partitura de voz o para implementar pistas de tipo karaoke.

Meta Evento	Tipo	Longitud	Datos
255 (0xFF)	(0x05)	Longitud de la cadena	Texto ASCII

Tabla 20. Valores del Meta Evento Lyrics.

Marker (Marcador)

Este evento marcar un momento en el tiempo significativo en la secuencia. Se suele encontrar el la primera pista, pero puede aparecer en cualquiera. Este evento puede ser muy útil a la hora de marcar el comienzo o final de estrofas o estribillos.

Meta Evento	Tipo	Longitud	Datos
255 (0xFF)	6 (0x06)	Longitud de la cadena	Texto ASCII

Tabla 21. Valores del Meta Evento Marcador.

Cue Point (Cuña)

Este meta evento marca el principio de un sonido o acción. Se suele encontrar en la primera pista aunque puede aparecer en cualquiera. En ocasiones es utilizado por secuenciadores para indicar el momento donde reproducir un sample o un video.

Meta Evento	Tipo	Longitud	Datos
255 (0xFF)	7 (0x07)	Longitud de la cadena	Texto ASCII

Tabla 22. Valores del Meta Evento Cuña.

MIDI Channel Prefix (Prefijo de Canal MIDI)

Este meta evento asocia los siguientes meta eventos a un canal MIDI. Su efecto termina con otro evento del mismo tipo o con cualquier evento que no sea meta evento. Comúnmente se utiliza antes del evento de nombre de un instrumento para especificar a qué canal representa tal nombre de instrumento.

Meta Evento	Tipo	Longitud	Datos
255 (0xFF)	32 (0x20)	1	0-15

Tabla 23. Valores del Meta Evento Prefijo de Canal.

End Of Track (Fin de Pista)

Este meta evento se utiliza para señalar el final de un paquete de pista y debe siempre aparecer como el último evento del paquete de esa pista.

Meta Evento	Tipo	Longitud
255 (0xFF)	47 (0x2F)	0

Tabla 24. Valores del Meta Evento Fin de Pista.

Set Tempo (Fijar Tempo)

Este meta evento especifica el tempo de la secuencia en términos de microsegundos por cuarto de nota. Está codificado en tres bytes. Normalmente se encuentra en el paquete de la primera pista, alineado en el tiempo para que ocurra en el mismo momento que una señal de reloj MIDI para conseguir mayor precisión. Si no se especifica un tempo, por defecto se toma 120 beats por minuto [McK]. La siguiente fórmula transforma el tempo de microsegundo por cuarto de nota a beats por minuto y viceversa.

```
MICROSECONDS_PER_MINUTE = 60000000

BPM = MICROSECONDS_PER_MINUTE / MPQN
MPQN = MICROSECONDS_PER_MINUTE / BPM
```

Código 1. Ecuaciones de conversión de tempo MIDI.

Meta Evento	Tipo	Longitud	Datos
255 (0xFF)	81 (0x51)	3	0-8355711

Tabla 25. Valores del Meta Evento Tempo.

SMPTE Offset (Offset SMPTE)

Este meta evento se utiliza para especificar el punto de comienzo de un offset SMPTE desde el principio de una pista. Se define en términos de hora, minutos, segundos, frames y sub-frames (siempre 100 por frame sin importar la subdivisión indicada en el paquete de cabecera MIDI). El byte utilizado para especificar el offset horario también especifica el frame rate en formato 0rrhhhhh, donde rr define el frame reate (00 = 14, 01 = 25, 10 = 30 con drop frame y 11 = 30) y hhhhhhh define la hora (0 a 23). El bit más significativo de la hora siempre está a 0. El rango posible del byte que define el frame depende del frame rate codificado en el byte de hora. Si se definió un frame rate de 25, el valor máximo será de 24.

Meta Evento	Tipo	Longitud	Hora	Min	Sec	Fr	SubFr
255 (0xFF)	84 (0x54)	5	0-23 *	0-59	0-59	0-30 *	0-99

Tabla 26. Valores del Meta Evento Offset SMPTE.

Time Signature (Definición del Compás)

Este evento sirve para establecer el compás de una secuencia. El compás se define con 4 bytes: un numerador, un denominador, un pulso de metrónomo y número de fusas por cuarto de nota. El numerador está indicado literalmente mientras que el denominador se especifica como el valor al cuál se debe elevar 2 para igualar el número de subdivisiones por negra. Por ejemplo; un valor de 0 significa que 2 elevado a 0 nos dará 1, una entera. Un valor de 1, dos elevado a 1 da 2, una corchea y así en adelante.

El pulso de metrónomo especifica cuán a menudo el metrónomo debería sonar en términos de número de señales de reloj por click, que tiene un período de 24 por cuarto de nota. Por ejemplo; un valor de 24 significaría sonar cada semicorchea (un beat) y un valor de 48 cada corchea (2 beats).

El último valor indica el número de fusas por cada 24 señales de reloj MIDI. El valor suele ser 8 pues normalmente hay 8 fusas en una corchea.

Al menos un evento time signature debería aparecer en el paquete de la primera pista (o de todas en fichero de tipo 2) antes de cualquier evento con delta time igual a cero. Por defecto se asume 4/4, 24, 8 si no se define otro.

Meta Evento	Tipo	Longitud	Numer	Denom	Metro	32nds
255 (0xFF)	88 (0x58)	4	0-255	0-255	0-255	01/01/55

Tabla 27. Valores del Meta Evento Compás.

Key Signature (Definición de la Clave)

Este meta evento se utiliza para especificar la clave y escala de una secuencia. Un valor positivo para la clave nos informa del número de sostenidos mientras que uno negativo el número de bemoles. Un valor de 0 para la escala indica mayor y 1 menor.

Meta Evento	Tipo	Longitud	Clave	Escala
255 (0xFF)	(0x59)	2	-7-7	0-1

Tabla 28. Valores del Meta Evento Clave.

Sequencer Specific (Específico de Secuenciador)

Este meta evento se utiliza para mandar información específica a un secuenciador hardware o software. El primer byte de datos (3 si el primero es 0) especifica el Id de fabricante y los siguientes bytes contienen información específica para cada fabricante.

Meta Evento	Tipo	Longitud	Datos
255 (0xFF)	127 (0x7F)	Longitud variable	Cualquier tipo y cantidad *

Tabla 29. Valores del Meta Evento Sequencer Specific.

System Exclusive Events (Eventos Exclusivos del Sistema)

También conocidos como SysEx. Esto eventos son utilizadas para controlar hardware MIDI o software que requiere datos especiales. Estos datos especiales son específicos de cada fabricantes. Todo evento SysEx incluye un Id que especifica a qué producto va destinado el mensaje, el resto de productos ignoran el mensaje. Hay tres tipos de eventos SysEx; para mandar datos en un solo

evento, a través de múltiples eventos, o para autorizar la transmisión de mensajes MIDI específicos.

Normal SysEx Events (Eventos SysEx Normales)

Estos son los más comunes. Se utilizan para mandar un bloque sencillo de información. El primer byte siempre es 0xF0 y el segundo es un valor de longitud variable que especifica la longitud de la información SysEx que le sigue. Los bytes de datos SysEx deben terminar siempre con 0xF7.

SysEx Event	Longitud	Datos
255 (0xFF)	Longitud variable	Bytes de datos, 0xF7

Tabla 30. Valores del Evento SysEx Normal.

Divided SysEx Events (Eventos SysEx Divididos)

Cuando la cantidad de información SysEx que se va a transmitir es muy grande, la transmisión de la totalidad en un solo mensaje podría hacer que el resto de eventos se transmitiesen después del momento de su reproducción.

Este tipo de mensajes permiten dividir la información en bloques que se transmiten separados para que el resto de eventos no sufran retardo.

El primer mensaje es exactamente igual a los anteriores salvo que no finaliza con 0xF7. El resto de eventos comenzarán con 0xF0 y el último finalizará con 0xF7.

Los eventos se separan en el tiempo para no congestionar el ancho de banda del protocolo MIDI.

SysEx Event	Longitud	Datos
240 (0xF0)	Longitud variable	Bytes de datos
247 (0xF7)	Longitud variable	Bytes de datos
247 (0xF7)	Longitud variable	Bytes de datos, 0xF7

Tabla 31. Valores del Evento SysEx Dividido.

Authorization SysEx Events (Eventos SysEx de Autorización)

El último tipo de eventos SysEx sirve para autorizar y permitir la transmisión de mensajes especiales. Estos pueden ser, por ejemplo, punteros de posición de una canción, selección de canciones o códigos de tiempo. Los eventos de este tipo se identifican por tener un valor de tipo de evento 0xF7.

SysEx Event	Longitud	Datos
247 (0xF7)	Longitud variable	Bytes de datos

Tabla 32. Valores del Evento SysEx de Autorización.

4.1.2 Planificación Automática

En planificación automática un problema queda definido por la descripción del estado de un entorno y de las metas que se pretenden satisfacer en ese entorno. El dominio contiene el modelo de todas las acciones que podemos realizar. El modelo de una acción expresa las condiciones que se deben cumplir para que una acción sea aplicable y los efectos que ésta tiene sobre el entorno.

El planificador por tanto se encarga de encontrar una sucesión de acciones tales que conviertan la descripción actual del entorno a otra que cumpla con las metas dadas.

Tanto el dominio como el problema tienen que estar representadas en un determinado lenguaje de representación. Una vez está definido, lo siguiente es decidir cuál va a ser el algoritmo de planificación que se va a utilizar. El algoritmo define el proceso de selección de la solución.

En la búsqueda del camino desde el estado inicial hasta la meta, el número de estados que se deben considerar es exponencialmente directo al tamaño del problema.

Tanto los algoritmos como los lenguajes de representación están directamente ligados con el modelo de planificación. El modelo de planificación determina básicamente las características del mundo (o de los problemas a resolver) que se van a tener en cuenta tanto en el lenguaje de representación como en el algoritmo. El modelo de planificación es más sencillo cuantas más simplificaciones se hagan respecto a qué es lo que se observa del mundo y cómo esto se representa.

4.1.2.1 Planificación Clásica

Es el modelo de representación más simple. Es por ello que mantiene unas condiciones muy restrictivas sobre la representación del mundo. Estas restricciones son las siguientes:

- Todas las acciones tienen un coste unitario.
- Las acciones no tienen duración; son transiciones instantáneas entre estados. El tiempo no está representado explícitamente.
- El sistema es completamente observable; en todo momento se conoce el estado de todo el sistema.
- Las acciones son deterministas; si una acción se puede aplicar a un estado, necesariamente lleva a un único estado.
- Se considera encontrado un plan cuando se han satisfecho todas las metas del problema de planificación.

4.1.2.1.1 Lenguajes de Planificación

En planificación clásica un problema se define como la tupla (P, A, I, G). P es el conjunto completo de proposiciones que se pueden dar en un estado. A es el conjunto de acciones. I es el estado inicial. G es el conjunto de proposiciones meta (una definición parcial de un estado).

Para un problema, una solución dijimos que era una secuencia de acciones, (a1, a2, a3, ... , an) A, cuya ejecución lleva a un estado en el cual todas las metas son ciertas.

Si dado un problema al menos existe una secuencia de acciones, el problema es resoluble. De todas las secuencias de acciones que pueden existir, aquella que tenga el menor número de acciones será óptima, es decir, será el plan óptimo.

La representación, actualmente se lleva casi toda a cabo en PDDL (Planning Description Domain Language). PDDL toma las características del lenguaje del planificador STRIPS y de ADL, así que primero daré una visión general sobre ambos para luego ejemplificar un ejemplo de PDDL.

Las siglas STRIPS responden a STandford Research Institue Problem Solver. En este lenguaje las acciones de un dominio se representan de la siguiente manera:

a = (pre(a), eff(a)+, eff(a)-)

las precondiciones de la acción, la lista de efectos que añade la ejecución (positivos) y la lista de

efectos que borra la ejecución (negativos).

Dado un determinado estado de un mundo s, la acción se podrá ejecutar si en el estado se cumplen las precondiciones de una acción, es decir:

pre(a) $\subseteq$ s

Si se cumple esta condición podemos ejecutar la acción a y el resultado es el siguiente:

resultado (s, a) = s U eff(a)+ - eff(a)-

El resultado de ejecutar una secuencia de acciones (a_1, ... , a_n) sobre un estado s se define recursivamente como:

resultado (s, (a_1, ... , a_n)) = resultado(resultado (s, (a_1, ... , a_{n-1})), a_n)

PDDL se propuso como lenguaje común para la competición internacional de planificación del año 1998 AIPS'98 Planning Competition. En la versión actual incluye tanto las características de STRIPS como ADL como extensiones propias. Las últimas versiones y sus añadido son los siguientes:

- PDDL 2.1 (2002): añade características relativas al manejo del tiempo y funciones objetivo (métricas).
- PDDL 2.2 (2004): añade predicados derivados y literales iniciados en el tiempo. Los primeros son axiomas encadenados hacia atrás que permiten al planificador conseguir hechos haciendo cierto el antecedente de alguno de ellos. Los segundos son literales que se hacen ciertos en un tiempo determinado, independientemente de lo que haga el planificador.

Los siguientes son ejemplos de un dominio y un problema para ese dominio específico expresados en PDDL.

```
;; "Travel" domain: A person has to get into a vehicle, drive it
;; somewhere, get out, and return to some other location.
;; I'm not sure where this domain comes from, probably along with
;; some UW planner (maybe UCPOP).

;; Note bug in domain: A vehicle becomes mobile when a person (driver)
;; enters it, and becomes immobile when the person disembarks. If two
;; persons enter a vehicle, it will become immobile when the first of
;; them disembarks.

(define (domain travel)
  (:requirements :strips)
  (:predicates
   (road ?from ?to) (at ?thing ?place) (driving ?p ?v) (bridge ?from ?to)
   (mobile ?thing) (person ?p) (vehicle ?v))

  (:action drive
    :parameters (?thing ?from ?to)
    :precondition (and (mobile ?thing) (road ?from ?to)
            (at ?thing ?from))
    :effect (and (at ?thing ?to) (not (at ?thing ?from))))

  (:action cross
    :parameters (?thing ?from ?to)
    :precondition (and (mobile ?thing) (bridge ?from ?to)
            (at ?thing ?from))
    :effect (and (at ?thing ?to) (not (at ?thing ?from))))

  (:action board
    :parameters (?person ?place ?vehicle)
    :precondition (and (person ?person) (vehicle ?vehicle)
            (at ?person ?place) (at ?vehicle ?place))
    :effect (and (driving ?person ?vehicle) (mobile ?vehicle)
      (not (at ?person ?place)) (not (mobile ?person))))

  (:action disembark
    :parameters (?person ?place ?vehicle)
    :precondition (and (person ?person) (vehicle ?vehicle)
            (driving ?person ?vehicle) (at ?vehicle ?place))
    :effect (and (at ?person ?place) (mobile ?person)
      (not (driving ?person ?vehicle)) (not (mobile ?vehicle))))
  )
```

Código 2. Ejemplo de dominio en PDDL.

```
(define (problem bulldozer1)
  (:domain travel)
  (:objects a b c d e f g jack bulldozer)
  (:init (at jack a) (at bulldozer e)
   (vehicle bulldozer)
   (mobile jack)
```

```
    (person jack)
    (road a b) (road b a)
    (road a e) (road e a)
    (road e b) (road b e)
    (road a c) (road c a)
    (road c b) (road b c)
    (bridge b d) (bridge d b)
    (bridge c f) (bridge f c)
    (road d f) (road f d)
    (road f g) (road g f)
    (road d g) (road g d))
  (:goal (and (at bulldozer g) (at jack a)))
 )
```

Código 3. Ejemplo de problema en PDDL.

4.1.3 MusicXML

Uno de los principales problemas de la distribución y edición de una pieza musical en formato escrito, es decir, una partitura, es que cada programa tiene una manera distinta de interpretar el formato MIDI. La forma en que el formato trata las notas, como eventos y no como entidades musicales, dificulta la edición en partitura. Cada programa de edición musical intentará plasmar sobre la partitura una cosa diferente, utilizando su propio algoritmo para traducir el MIDI en la partitura.

Este hecho conlleva el riesgo de que cuando se realiza un trabajo sobre una pieza con distintos software, existe la posibilidad de perder, al cambiar entre programas, los arreglos sobre esta representación.

Mientras que en el plano musical, los programas se entienden dado que el estándar MIDI lo permite. No existe manera de expresar en ese estándar informaciones sobre las notaciones o la colocación en la partitura de la pieza. En este marco es donde se han desarrollado varios intentos de estándar para paliar las limitaciones de representación que contenía MIDI [Cun].

Estos nuevos estándares no intentan sustituir MIDI como formato de representación musical. Su intención es la de crear un estándar que se encargue de asegurar una representación de la notación de esa música de manera homogénea entre los distintos programas.

De las numerosas representaciones que existen, algunas tienen en común el uso de XML como medio expresivo. Algunas de estas son WEDELMUSIC, MUTATED, XCHORDS y por último MusicXML.

Esta última iniciativa es la más madura, principalmente por haber sido desarrollada por una empresa comercial conjunta y paralelamente con un software de uso extendido. A la par las licencias son libres lo cual asegura que surgirá software libre de terceras partes que soporte este formato. Otra razón importante para su adopción en primer lugar es que su estructuración de la información musical es exactamente la misma que adoptamos desde un inicio en este proyecto.

MusicXML es un formato de ficheros de música basado en XML. Es un proyecto abierto. Se ha desarrollado por Recordare. El estándar MusicXML está definido por una serie de definiciones de tipo de documento (DTD) libremente redistribuibles bajo la licencia pública de MusicXML DTD [Exp].

Las principales metas de MusicXML son proponer un estándar de intercambio y representación de archivos musicales Internet-friendly y que no fuese en formato binario. Estas dos características apuntaban a la posible aplicación de XML como plantilla. La adopción de XML significa crear un

formato con un alto nivel de legibilidad humana y con herramientas de apoyo existentes.

MusicXML está basado en las normas y formatos definidos en el meta-lenguaje XML. XML es un subconjunto de SGML (Standard generalizadas Mark-up Language). Los documentos XML son contenedores de información, MusicXML ha sido diseñado para la representación de música en partitura. Prevé el almacenamiento digital, así como el intercambio de los datos musicales.

La creación de un documento XML permite la separación de los datos contenidos, así como una definición de la estructura de archivos, controlada por la creación y referenciación a los DTD. Este mecanismo de definición de la información sobre otro tipo de información es la razón por la cual el XML se denomina a menudo un meta-lenguaje. MusicXML emplea actualmente a los archivos DTD como un mecanismo de control de documentos, en contraposición al esquema XML.

4.1.3.1 Estructura

Esta sección cubre los principales elementos que están definidos en el estándar. Cada fichero xml está dividido inevitablemente en dos partes. Por un lado, dentro del tag <PART-LIST> encontramos un resumen de las partes siguientes que contendrán la notación. Esto es necesario para introducir los elementos que más tarde serán leídos.

A continuación de esta introducción, vamos a encontrar las partes. Cada una va rodeada de los tags <PART> e identificada por un id unívoco.

Para cada parte, encontramos todos los compases enunciados como medidas (<meassures>). Están numeradas con el atributo xml number y esto permite ordenarlas en el tiempo y relacionar varias partes entre ellas.

Encontramos aquí dentro ya elementos relevantes para la maquetación. La clave y la llave (<clef>, <key>), no aportan ninguna información musical. Por otra parte dentro de <attributes> tenemos los siguientes tags dándonos información relevante:

- <division>. Dictamina en cuantas porciones de tiempo (ver debajo) dividimos las duraciones.
- <time>. Es el compás. <beats> entre <beats-type>

Salimos de la definición de atributos y encontramos ya los elementos musicales. Todo está expresado como notas. El tag <note> tiene anidado dentro los siguientes tags de los cuales extraeremos la información musical de cada una de las composiciones:

- <pitch>. La altura de la nota.
- <duration>. La duración de la nota expresada en función de divisions.
- <type>. Información meramente maquetadora sobre el carácter que representará la nota.

4.1.3.2 Ejemplo

El siguiente xml contiene los elementos antes descritos. Incluye únicamente una nota envuelto en los tags principales.

```
<!DOCTYPE score-partwise PUBLIC "-//Recordare//DTD MusicXML 0.6 Partwise//
EN" "http://www.MusicXML.org/dtds/partwise.dtd">
<score-partwise>
     <part-list>
          <score-part id="P1">
               <part-name>Music</part-name>
          </score-part>
     </part-list>
     <part id="P1">
          <measure number="1">
               <attributes>
                  <divisions>1</divisions>
                  <key>
                     <fifths>0</fifths>
                  </key>
                  <time>
                     <beats>4</beats>
                     <beat-type>4</beat-type>
                  </time>
                  <clef>
                     <sign>G</sign>
                     <line>2</line>
                  </clef>
               </attributes>
               <note>
                  <pitch>
                     <step>C</step>
                     <octave>4</octave>
                  </pitch>
                  <duration>4</duration>
                  <type>whole</type>
               </note>
          </measure>
     </part>
</score-partwise>
```

Código 4. Ejemplo de formato MusicXML.

4.1.4 Java

Java es un lenguaje de programación orientado a objetos de código interpretado. Esto lo hace multiplataforma, portable sin recompilación. Ha sido el elegido para programar el creador de dominios, la interfaz y todos los módulos que ésta utiliza.

En esta sección explicaré las librerías utilizadas para el parseo de xml, la generación de ficheros en formato MIDI y el acceso a base de datos.

4.1.4.1 MIDI en Java

La librería javax.sound, la api de sonido de java, contiene una serie de clases que permiten la implementación de aplicaciones enfocadas al uso de sonidos grabados y sintetizados. Los sonidos sintetizados son, como se indicó en la introducción, producidos por máquinas que están casi siempre controlados por MIDI.

Es por ello que dentro de la sección de sintetización de sonidos, a parte de la generación del sonido como tal, existen clases para controlar estos por MIDI y tratar ficheros en su formato. Esto nos permite generar ficheros MIDI.

Si sumamos a este hecho, la orientación a objetos de Java, podemos conseguir una traducción a

ficheros MIDI desde una estructura de objetos que defina los términos musicales que nos atañen.

Las principales características que las librerías de Java nos ofrecen son las siguientes:

- Implementación de los mensajes MIDI a través de las clases:

  javax.sound.midi.ShortMessage

- Implementación de los eventos MIDI a través de las clases:

  javax.sound.midi.Event

- Implementación de Secuencias y Pistas a través de las clases:

  javax.sound.midi.System

  javax.sound.midi.Sequece

  javax.sound.midi.Track

- Implementación de Entrada/Salida de fichero a través de las clases:

  java.io.file

4.1.4.2 XML en Java

Para el procesamiento de XML con Java, Java ofrece SAX ("Simple Api for XML"). SAX es una librería que permite el procesamiento de la entrada por eventos. Esto significa que procesa cada uno de los tags y dispara acciones según estos se vayan abriendo y cerrando.

De todas las posibilidades que ofrece la librería java.xml, tan sólo utilizaremos lo anterior para procesar la entrada en formato MusicXML y realizar una serie de acciones en función de los tags que contenga el fichero.

4.1.4.3 MySQL en Java

Cuando se desea acceder a una base de datos desde las clases de java, se utiliza un conector. Un conector es una librería (en Java en este caso) que nos da una serie de objetos/métodos a través de los cuales se pueden hacer operaciones sobre la base de datos que queramos.

Las operaciones son sentencias en SQL. Nuestra base de datos es compatible con SQL, queriendo decir con esto, que las operaciones que realizamos sobre ella son únicamente con sentencias compatibles con este lenguaje aunque la base de datos sea otra.

Particularmente, la implementación que utilizamos es MySQL, y por tanto el conector debe ser para esta base de datos. MySQL ofrece el conector jdbc para java. JDBC responde a "Java Data Base Connector".

Los contenidos a los que deseamos acceder son una base de acordes cuya estructura e información almacenada se describe en detalle en la sección [4.2.3.1].

4.1.5 Teoría Musical

Existen múltiples definiciones para la música. Por un lado podemos explicarla como una sonoridad organizada, teniendo una formulación perceptible, coherente y significativa; y a razón de ello, se puede expresar en unos términos bien definidos.

Por otra parte, atendiendo a definiciones clásicas, podemos argumentar que la música es un

conjunto de sonidos y silencios organizados de manera vertical y horizontal. A esta verticalidad y horizontalidad las llamamos respectivamente armonía y melodía.

4.1.5.1 Cualidades del sonido

Podemos diferenciar un sonido de otro fijándonos en sus parámetros principales, la altura, duración, intensidad y timbre[Her].

4.1.5.1.1 Altura

La altura es una respuesta en frecuencia. Normalmente las frecuencias de los sonidos están clasificadas. Esta clasificación son los nombres de las notas. Cada nombre tiene una longitud de onda. Además, podemos, a groso modo, diferenciar entre tonos graves y agudos.

4.1.5.1.2 Duración

La duración de una nota se refiere al tiempo que una nota está sonando. El tiempo que pasa desde que una nota empieza a sonar hasta que termina de hacerlo.

4.1.5.1.3 Intensidad

La intensidad, por otra parte, denota lo que comúnmente entendemos por volumen. Es la fuerza con la que se produce un sonido.

4.1.5.1.4 Timbre

El timbre es la cualidad de un sonido que permite diferenciarlo del resto. Ya que los sonidos, normalmente no son una onda simple, sino que son una composición de ellas, el timbre es la diferencia, por ejemplo, del sonido de una guitarra y de una flauta, aunque todas las cualidades anteriores sean iguales; estén sonando con el mismo volumen, el mismo tiempo en la misma nota.

4.1.5.2 Armonía musical

Un modo se define como la manera de distribuir las notas de una escala en una determinada sucesión de intervalos de tono o semitono.

Se usan principalmente dos modos, el mayor y el menor. Éstos provienen de los modos gregorianos, jónico y eolio, aunque de menor uso también pueden emplearse los modos dórico, frigio y mixolidio. Los modos lidio y locrio son de raro uso ya que no contienen uno de los grados tonales.

El modo mayor es el modo en el cual está escrita la mayoría de la música moderna y por tanto es tal el que acapara toda la atención.

El modo mayor (probablemente se podría decir arriba que tratamos la armonía y melodía desde el punto de vista del modo mayor).

4.1.5.2.1 Notas diatónicas y cromáticas

Las notas diatónicas son siete y vienen definidas por los nombres naturales Do, Re, Mi, Fa, Sol, Si, Do, con las alteraciones adecuadas según sea el centro tonal escogido.

El resto de notas son cromáticas, hay que notar que hay cinco sonidos cromáticos y diez notas

cromáticas debido a la enarmonía.

A las siete notas diatónicas se les llama grados y tienen además un nombre propio cada uno.

I	tónica
II	supertónica
III	mediante
IV	subdominante
V	dominante
VI	superdominante
VII	Sensible

Tabla 33 . Nombre de los grados de la escala.

4.1.5.2.2 Melodía diatónica

Una melodía diatónica será la formada únicamente por grados diatónicos a una escala determinada.

4.1.5.2.3 Armonía diatónica

Como armonía entendemos la parte de la música que trata el estudio de los sonidos simultáneos, así, armonía diatónica será la formada por grupos de sonidos diatónicos.

4.1.5.2.4 Acordes tríadas

Son grupos de tres sonidos y se forman superponiendo dos terceras diatónicas consecutivas; sobre cada grado de la escala podremos formar un acorde tríada.

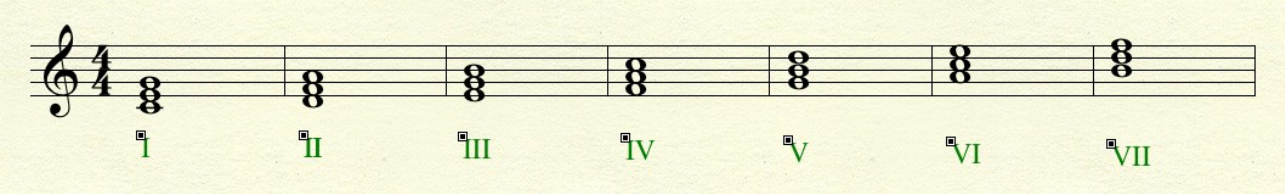


Ilustración 1. Acordes tríadas en tono Do.

El grado sobre el que se forma un determinado acorde se llama fundamental, la primera tercera superpuesta se le llama “tercera del acorde” y a la segunda tercera: “quinta del acorde”, por su intervalo de quinta que forma con la fundamental.

4.1.5.2.4.1 Clasificación de los acordes tríadas

Los siete acordes que se forman sobre los siete grados de la escala no son iguales, ya que los intervalos entre la fundamental y su tercera y su quinta no lo son, ello motiva la clasificación por especies de los mismos.

Los acordes que se forman sobre la escala mayor son:

sobre el I	grado	3ª mayor	5ª justa
sobre el II	grado	3ª menor	5ª justa

sobre el III	grado	3ª menor	5ª justa
sobre el IV	grado	3ª mayor	5ª justa
sobre el V	grado	3ª mayor	5ª justa
sobre el VI	grado	3ª menor	5ª justa
sobre el VII	grado	3ª menor	5ª disminuida

Tabla 34 . Lista de acordes de la escala mayor.

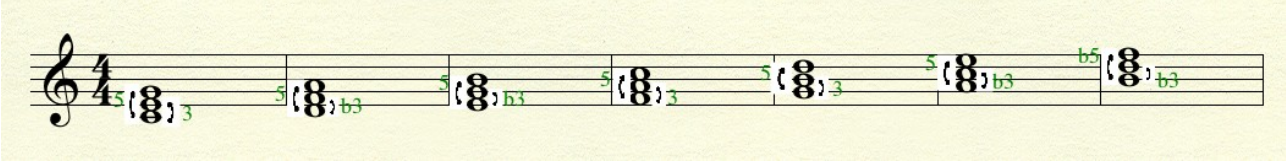


Ilustración 2. Intervalos en los acordes de la escala mayor.

Estos acordes se pueden reunir en tres grupos: los que tienen tercera mayor y quinta justa que se llaman “mayores” y los de tercera menor y quinta justa que se llaman “menores” y los de tercera menor y quinta disminuida que se llaman “disminuidos”.

Una vez agrupados vemos que los acordes formados sobre el I, IV y V grado son mayores, los formados sobre el II, III y VI grados son menores y el construido sobre el VII es disminuido.

Acordes	especie	fórmula
I,IV,V	Mayor	1,3,5
II, III, VI	Menor	1,b3,5
VII	Disminuido	1,b3,b5

Tabla 35 . Clasificación de acordes según mayores, menores y disminuido.

Los acordes que se forman sobre una determinada escala mayor no son exclusivos de ésta, así vemos cómo el acorde que se forma sobre el primer grado de la escala del tono de Do, es el mismo que se forma sobre el IV grado de la escala de Sol o el mismo que se forma sobre el V grado del tono de Fa. O sea, que todo acorde mayor se forma en tres escalas diferentes.

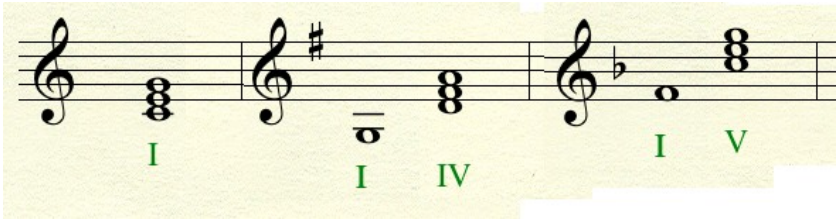


Ilustración 3. Acorde de Do en tres escalas distintas.

A un acorde menor le sucede lo mismo, así el acorde del II grado del de Do será el mismo que que se forma sobre el III de Si bemol, o el VI de Fa.

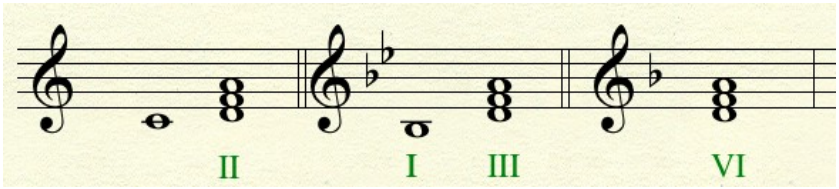


Ilustración 4. Acorde de Re Menor en tres escalas distintas.

En cambio el acorde de VII grado es único y sólo se forma en una escala.

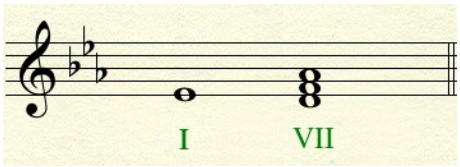


Ilustración 5. Acorde de Séptima disminuido en escala de Do.

4.1.5.2.4.2 Inversión de acordes

Existen tres estados en los que se pueden encontrar los acordes de tríada:

4.1.5.2.4.2.1 Estado fundamental

Cuando un acorde tiene la fundamental como nota más grave, ésta está en estado fundamental, cuando la nota más grave no es la fundamental, se dice que el acorde está invertido.

4.1.5.2.4.2.2 Primera inversión

Está en primera inversión un acorde, cuando tiene como nota más grave a su tercera.

4.1.5.2.4.2.3 Segunda inversión

Está en segunda inversión un acorde cuando tiene como nota más grave a su quinta.

4.1.5.2.5 Progresión armónica

Una progresión armónica es una sucesión de acordes.

4.1.5.2.5.1 Armonización de una melodía nota a nota

Cada nota melódica puede ser armonizada con tres acordes diatónicos, ya que ésta puede ser considerada fundamental, 3ª o 5ª de un acorde.

Se debe elegir un acorde y colocarlo en una disposición en la que la primera voz sea la nota melódica a armonizar.

La siguiente melodía podría armonizarse de la siguiente forma:

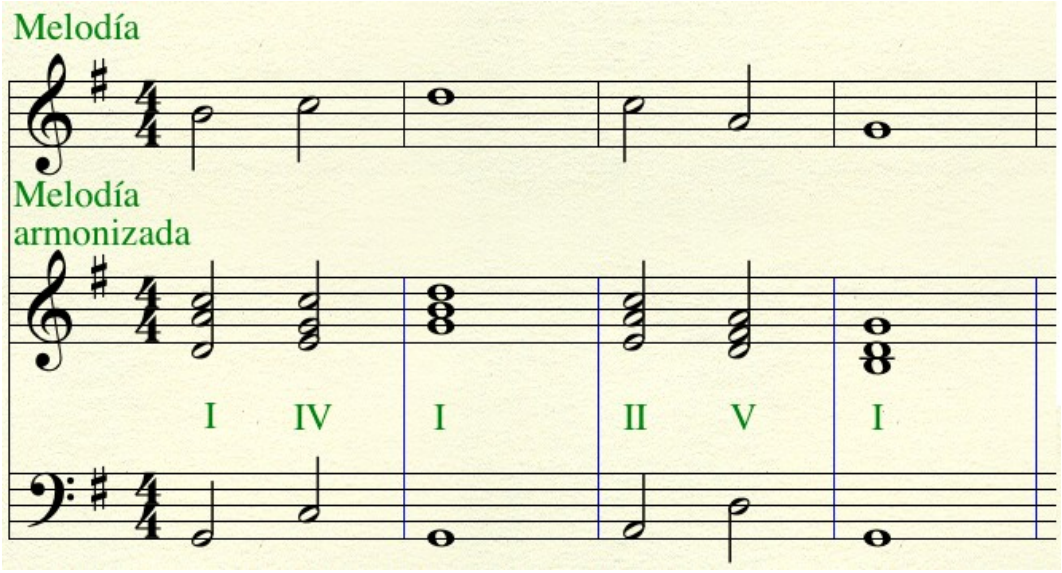


Ilustración 6. Ejemplo de armonización nota a nota.

En una misma composición puede aparecer la misma nota a distinta altura, por ende, este fenómeno se puede repetir en la armonización.4

4.2 Music Maker

En esta sección se describe cada una de las partes de las que se ha compuesto el desarrollo. Las partes contienen el desarrollo teórico previo a la codificación y la consecución práctica. Claramente, la distinción entre estas etapas de todo desarrollo no es clara y de la implementación práctica se retorna al estudio teórico debido a dificultades que primeramente no se consideraban.

Primeramente se describe el dominio, objetivo principal del proyecto. La correspondencia entre el dominio y el formato MIDI (para exportación) y MusicXML (para importación) son tratados después. Por último se aborda el formato de la base de datos y el interfaz gráfico de usuario (GUI).

4.2.1 Acompañamiento de melodías mediante planificación Automática

En la descripción del dominio, nuestro pequeño mundo, hay tres partes que forman parte de la planificación y una más que hace falta para generar acordes. A continuación se describen estas cuatro partes.

4.2.1.1 Dominio

El dominio representa la descripción del mundo sobre el que se pretende actuar. En primer lugar se debe definir los tipos de datos que vamos a tener. También se ha de definir las constantes, los predicados y, por último, las acciones.

La conjunción de todo esto será lo que permita transformar la definición inicial del problema en el objetivo final.

En la definición de los tipos de datos, todos los tipos heredan del tipo object. Se tiene pues una jerarquía de tipos. En el primer nivel se contemplan pitch, octave, note y acorde.

Pitch y Octave representan el valor musical de una nota, es decir, qué nota es. Los valores que

toman no son sin embargo el nombre de las notas, son tonos relativos a la nota tónica de la melodía. Esto es definido en el momento de la creación del problema/melodía.

Acorde es un tipo que se utiliza para acompañar una nota.

Note puede existir de tipo abstracto y normal (abstract/normal) y este es otro nivel en la jerarquía. De tipo abstracto se tienen start y finish. Éstas son la primera y la última nota y están ahí para definir por dónde empezar a acompañar y por donde terminar. Normal son el resto de notas que sí forman la melodía.

Esta porción siguiente de código forma parte del dominio y es donde está definido el tipado anterior:

```
  (:types

     start finish - abstract
     abstract normal - note
     pitch octave note acorde - object

  );; end types
```

Código 5. Definición de tipos.

Las constantes definidas nos permiten dar valor a los tipos de datos anteriores. Esto se va a conseguir con ayuda de los predicados que se explican más adelante. Primero veamos cómo se definen en un dominio en PDDL:

```
(:constants
  ;;ocho octavas
  O1 O2 O3 O4 O5 O6 O7 O8 - octave
  ;;Las notas y el silencio
  p1 p1s p1f p2 p2s p2f p3 p3s p3f p4 p4s p4f p5 p5s p5f p6 p6s p6f p7 p7s
p7f s - pitch
  ;;Los acordes
  I I2 II II2 III III2 IV IV2 V V2 VI VI2 VII VII2 S  - acorde )
;;end of constants
```

Código 6.Definición de constantes.

En primer lugar están las octavas. Hay ocho definidas en este dominio. La denominación que se ha tomado es la de una vocal O mayúscula seguida del número de la octava. A continuación se tienen las alturas. Su denominación es relativa a la tónica de la melodía. El nombre es autoincremental, es decir, p2 es el tono superior a p1 hasta llegar a p7 donde se volvería a empezar a contar aumentando la octava. Entre cada dos tonos existen semitonos. Aunque sólo se contempla un semitono, se incluyen ambas denominaciones en el dominio, la del tono superior y la del inferior. Esto significa que p1s (del inglés sharp) es el tono primero sostenido y es equivalente al segundo bemól, p2f (del inglés flat).

Gráficamente, sobre el teclado de un piano, el concepto de alturas y octavas queda expresado de la siguiente manera:

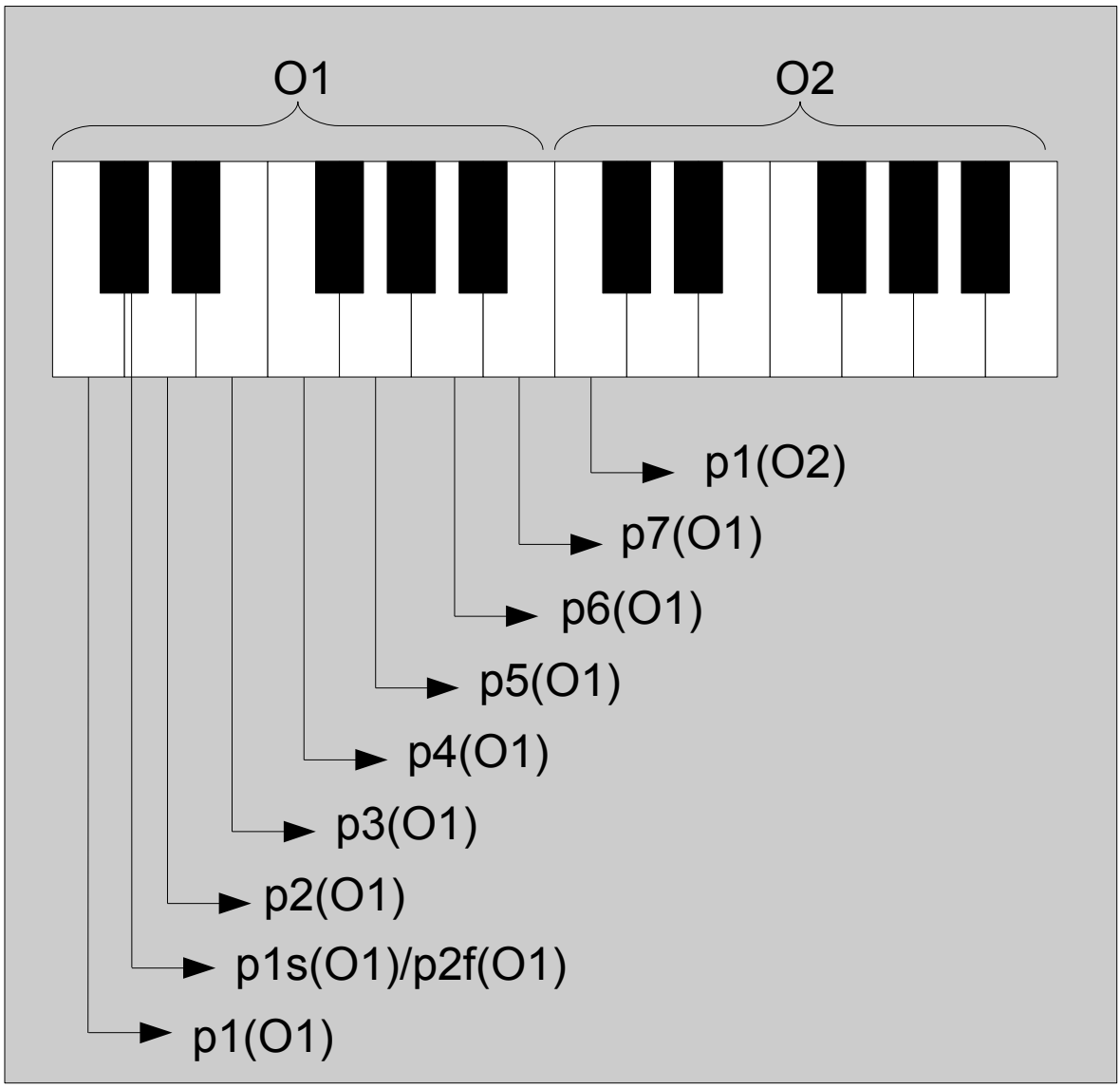


Ilustración 7. Relación teclado del piano pitch/octave.

Los acordes tienen un nombrado libre. Si la creación del dominio se hace con la interfaz gráfica desde la base de datos, el nombre corresponderá con el campo nombre de la tabla chord. Todos los acordes que se vayan a utilizar son definidos aquí colocando su nombre.

La siguiente sección (predicates) nos ayuda en tres puntos:

- Enunciar el predicado que permite dar a cada nota un pitch y un octave. La forma de hacerlo será con el predicado:

```
     ;;the pitch of a note is defined with the pitch (constant) and the
octave (constant)
     ;; only a normal note will have pitch
     ;;definimos la altura de una nota con el nombre de su nota(constante)
y la octava (constate)
     ;; sólo las notas que son normales tienen altura
     (pitch ?n - note ?p - pitch)
     (octave ?n - note ?o - octave)
```

Código 7. Definición de pitch/octave en MusicMaker.

- Definir el predicado que permita indicar con qué acorde se ha de acompañar una nota determinada.

```
     ;;This action sets the chord to go with a note
     ;;Esta acción acompaña a una nota con un acorde
     (acomp ?n1 - note ?chord - acorde)
```

Código 8. Definición de acorde de acompañamiento en MusicMaker.

- Predicar cuando una nota está acompañada:

```
     ;;This predicate tells that a certain note has already been
accompanied
     ;;Este predicado nos indica que cierta nota ya ha sido acompañada
     (accompanied ?n - note)
```

Código 9. Definición de nota acompañada en MusicMaker.

- Indicar la secuenciación de las notas.

```
     ;;the precedence of the notes is defined with this predicate. Every
note has a next one except for the last one, finish.
     ;;El orden de las notas se define mediante esta función. Toda nota
tiene siguiente a excepción de la última, finish.
     (next-note ?n1 ?n2 - note)
```

Código 10. Definición de secuencia de notas en MusicMaker.

La distinción entre los predicados y las funciones es que las segundas permiten dar valores numéricos. Es por ello que son utilizadas para definir la longitud de una nota, el compás en el que se encuentra, su posición dentro de éste y la velocidad.

```
  (:functions
    ;;Length of a note
    ;;Duración de la nota : 16 = negra, 1 = semifusa, 64 = redonda
    (length ?n - normal)

    ;;Position in a compass in which a note starts
    ;;Posición en el compás en la que una nota empieza
    (init ?n - note)

    ;;velocity of the note
    ;;velocidad de la nota
    (speed ?n - normal)

    ;;compass of the note
    ;;compás al que pertenece una nota
    (comp ?n - normal)
  );;end functions
```

Código 11. Definición de funciones de MusicMaker.

El último bloque de definiciones de un dominio son las acciones. Además de las dos acciones comenzar y terminar, va a existir una acción por cada tono que pueda disparar un acorde, por cada uno de estos acordes. Si existen tres acordes de los cuales cada uno puede ser colocado con dos tonos distintos, en total se tienen seis acciones para definir esta casuística en concreto.

Las acciones comenzar y terminar:

```
  (:action comenzar
    ;; n1 es la nota start
    ;; n2 es la nota que va después
    :parameters (?n1 - start ?n2 - note)

    :precondition
       (next-note ?n1 ?n2) ;; tiene que haber una nota entre medias por
lo menos!!!!
    :effect
       (accompanied ?n1)
  );;fin comenzar

  (:action finalizar
    ;; n1 es la nota finish
    ;; n2 es la nota que va antes
    :parameters (?n1 - finish ?n2 - note)

    :precondition
       (and
          (next-note ?n2 ?n1)
          (accompanied ?n2)
       )
    :effect
       (accompanied ?n1)
       ;;nos bastaría con acompañar a la última nota para saber que hemos
terminado.
  );;fin finalizar
```

Código 12. Definición de acciones de MusicMaker.

Estas acciones tienen una propiedad particular cada una y es el tipo de notas que toman como parámetros. Si se observa comenzar, se ve que recibe una nota de tipo start. Como la nota de tipo start es la primera y la secuencia de la melodía va seguida del resto, sucede que en el estado inicial del problema, se cumplen las precondiciones de esta acción y se llevará a cabo, por lo tanto se acompañará a la nota start. No tendrá un acorde relacionado, pero estará acompañada.

En cuanto a la acción terminar, sucede que recibe de parámetros una nota del tipo finish. La nota finish se coloca después de la última de la melodía. Cuando se cumplan las precondiciones de esta acción se habrán acompañado todas las notas anteriores y entonces se dará por acompañada (sin acorde de nuevo) esta última. Si se define como meta del problema que esta última nota esté acompañada, se habrá acompañado todas al llegar a la meta.

Además de las dos acciones anteriores, ya se explicó que cada acorde tiene una serie de acciones relacionadas. La estructura que sigue cada uno de los disparadores de un acorde es la siguiente:

```
;;acorde: I, acción: 0
(:action I_0
   ;; n1 es la nota actual
   ;; n2 es la nota previa
   :parameters (?n1 ?n2 - note)

   :precondition
      (and
         (pitch ?n1 p1)
                              (and
                                  (accompanied ?n2)
                                  (next-note ?n2 ?n1)
                              )
      )
   :effect
      (and
         (acomp ?n1 I)
         (accompanied ?n1)
      )
)
```

Código 13. Definición de una acción de acorde de MusicMaker.

Las acciones son nominadas por el nombre del acorde, barra baja y un número. Como parámetro recibe las dos notas para comprobar la secuencia (actual y anterior). Las precondiciones que se tienen que dar son la secuenciación, que la anterior esté en efecto acompañada y que el tono de la actual sea el mismo que el del disparador.

Como efecto, se coloca el acorde I (pues sobre este es el ejemplo) a la nota y la marcaremos como acompañada.

4.2.1.2 Problema

Un problema es una descripción del mundo inicial y una meta, es decir, una serie de condiciones que queremos llegar a cumplir.

En la descripción del dominio ya se han explicado todas las herramientas que se necesitan para describir un estado del mundo y poder modificarlo. La descripción de un problema por tanto se valdrá de estas herramientas.

El problema define la melodía que se pretende acompañar. Su descripción es la descripción de cada una de las notas que la componen. También debemos expresar la secuenciación de las notas. Se añade a lo anterior las notas de principio y fin y la meta y resulta el fichero del problema.

El ejemplo más fácil de secuenciación es el que contiene dos notas. En el código siguiente se pueden ver dos notas con todas sus propiedades definidas y el predicado de su secuenciamiento:

```
(pitch n1 s)
(octave n1 O4 )
(= (speed n1) 0)
(= (length n1) 16)
(= (init n1) 0)
(= (comp n1) 1)

(next-note n1 n2)

(pitch n2 p3)
(octave n2 O4 )
(= (speed n2) 0)
(= (length n2) 16)
(= (init n2) 16)
(= (comp n2) 1)
```

Código 14. Ejemplo de secuenciación de notas de MusicMaker.

Sin embargo, lo anterior es sólo una porción de un fichero completo. Los bloques que componen una descripción de problema completa son la declaración de los objetos, el estado inicial y las metas. Para que tenga sentido según las restricciones de nuestro dominio, la primera nota debería ser start y la última finish. La meta será que final esté siempre acompañada. Una versión completa de la porción anterior sería la siguiente:

```
(define (problem SONG)

   (:domain armonia)

      (:objects
        n0 -start
        n1 n2 - normal
        nlast - finish
      )
   (:init
      (pitch n1 s)
      (octave n1 O4 )
      (= (speed n1) 0)
      (= (length n1) 16)
      (= (init n1) 0)
      (= (comp n1) 1)

      (pitch n2 p3)
      (octave n2 O4 )
      (= (speed n2) 0)
      (= (length n2) 16)
      (= (init n2) 16)
      (= (comp n2) 1)

      (next-note n0 n1)
      (next-note n1 n2)
      (next-note n2 nlast)

      (accompanied n0)
   )

   (:goal     ;;aquí es donde digo que finish este acompañada
      (accompanied nlast)
   )

);;end of define
```

Código 15. Ejemplo de problema en PDDL para MusicMaker.

En la descripción inicial del problema, start está acompañada, para que se siga acompañando por la primera de la melodía, segunda en la descripción del problema.

4.2.1.3 Planificador

LPG (Local Search for Planning Graphs) es un planificador basado en búsqueda local y planificación de grafos que maneja dominios descritos en PDDL2.1 que incluyan cantidades numéricas y duraciones. El sistema puede resolver problemas tanto de generación de planes como de adaptación. El sistema de búsqueda de LPG se inspiró en Walksat, un procedimiento eficiente para resolver problemas SAT.

El espacio de búsqueda de LPG se compone de "grafos de acción", subgrafos del grafo de planificación que representan planes parciales. Cada uno de los pasos de la búsqueda significa una modificación del grafo que convierte un grafo de acción en otro. LPG explota una representación compacta del grafo de planificación para definir la vecindad de la búsqueda y evaluar sus elementos utilizando una función parametrizada en la cual los parámetros sopesan diferentes tipos de inconsistencias en el plan parcial actual y son dinámicamente evaluados durante la búsqueda utilizando multiplicadores discretos de Lagrange.

La función de evaluación utiliza heurísticas para estimar el coste total y el coste de ejecución para alcanzar una precondición. La duración de las acciones y los valores numéricos son representados

en los grafos de acción y son evaluados en la función de evaluación. En dominios temporales las acciones son ordenadas utilizando un grafo de precedencia que se mantiene durante la búsqueda y tiene en cuenta las relaciones mutex del grafo de planificación [Lpg].

LPG es un gran candidato a este proyecto por su búsqueda local estocástica. A diferencia de otros planificadores, como FF, no es determinista, puede generar diferentes soluciones para un mismo problema en distintos momentos. Ese componente de aleatoriedad es la principal razón de su uso. Se consigue obtener distintas planificaciones para una misma melodía con distintas ejecuciones del planificador.

4.2.1.4 Solución

La solución es la salida del planificador. El planificador genera un plan, una secuencia de acciones que se deben llevar a cabo. Cada una de las acciones representa acompañar a una nota con un acorde. Ya que no se puede ver la descripción del problema en cada cambio, lo cual sería ideal para comprobar el predicado acomp y extraer el acorde, se debe decodificar de la salida.

El fichero de plan puede tener lineas comentadas que comienzan por “;”. Éstas las ignoramos. El resto de entradas del fichero tienen este formato:

```
0:   (S_0 N1 N0) [1]
```

Código 16. Formato de salida del planificador.

El primer número significa el número de acción. Después entre paréntesis se encuentra la acción que hemos llevado a cabo y al final entre corchetes el coste de esta. Como no se está ejecutando una planificación con costes esto es 1 siempre.

Si se observa el contenido entre paréntesis; se tiene el nombre del acorde antes de la barra baja. La segunda palabra es el nombre de la nota que estamos acompañando. Esta información es toda la que se va a necesitar para poder generar un acompañamiento: qué acorde poner a qué nota.

El siguiente ejemplo muestra una salida más elaborada que el anterior acompañamiento:

```
; Version LPG-td-1.0
; Seed 60926154
; Command line: lpg-td/lpg-td-1.0 -o /home/cr/Escritorio/Clasico/domain
-inst_with_contraddicting_objects -f /home/cr/Escritorio/pruebas/bach/bach
-speed -out /home/cr/Escritorio/pruebas/bach/Clasico1/clasico-plan
; Problem /home/cr/Escritorio/pruebas/bach/bach
; Actions having STRIPS duration
; Time 0.15
; Search time 0.09
; Parsing time 0.04
; Mutex time 0.01
; MetricValue 63.00

0:   (I_1 N1 N0) [1]
1:   (IV_1 N2 N1) [1]
2:   (I_2 N3 N2) [1]
3:   (I_2 N4 N3) [1]
4:   (IV_1 N5 N4) [1]
5:   (S_2 N6 N5) [1]
6:   (II_0 N7 N6) [1]
7:   (S_1 N8 N7) [1]
8:   (I_1 N9 N8) [1]
9:   (FINALIZAR NLAST N9) [1]
```

Código 17. Formato completo de salida del planificador.

4.2.1.5 Fichero de acordes

El fichero de acordes contiene una descripción de cada uno de los acordes para utilizarla más tarde a la hora de generar un acorde como notas.

Por cada acorde se tendrá una línea especificando su nombre siguiendo a un punto (para diferenciar el comienzo de un acorde). Después en las siguientes lineas hasta el siguiente acorde se encuentran las notas que forman el acorde de manera relativa. Como ya se explicó anteriormente esto quiere decir que se encuentran definidos el tono relativo (p1, p2, p3, ...) seguido de la octava de la nota.

La siguiente es una definición del acorde primero mayor:

```
.I
p1,O3
p3,O3
p5,O3
```

Código 18. Formato del fichero de acordes.

4.2.2 Correspondencia MusicXML problema

A la hora de transformar el fichero MusicXML en un fichero de problema PDDL, no todos los elementos del primer formato son útiles para el segundo. Viceversa, no todos los elementos que necesita un fichero de problema están disponibles en el segundo.

Esto significa que la correspondencia no es 1 a 1. También establece la manera en que se debe parsear el fichero xml. En la tabla siguiente se pueden observar los elementos de MusicXML que encuentran correspondencia en la definición del problema sobre un ejemplo:

MusicXML	Problem Definition
<!DOCTYPE score-partwise PUBLIC "-//Recordare//DTD MusicXML 0.6 Partwise//EN" " http://www.MusicXML.org/dtds/partwise.dtd ">	

<score-partwise>	
<part-list>	
<score-part id="P1">	
<part-name>Music</part-name>	
</score-part>	
</part-list>	
<part id="P1">	
<measure number="1">	(= (comp <nota>) 1) para todas las notas del compás
<attributes>	
<divisions>1</divisions>	
<key>	
<fifths>0</fifths>	
</key>	
<time>	
<beats>4</beats>	
<beat-type>4</beat-type>	
</time>	
<clef>	
<sign>G</sign>	
<line>2</line>	
</clef>	
</attributes>	
<note>	
<pitch>	
<step>C</step>	(pitch n1 p1)
<octave>4</octave>	(octave n1 O4)
</pitch>	
<duration>4</duration>	(= (length n1) 64)
<type>whole</type>	
</note>	
</measure>	
</part>	
</score-partwise>	

Tabla 36. Relación MusicXML – Definición del Problema.

Como se puede observar la cantidad de elementos que se mapea es muy pequeña. Del código xml se deben hacer las siguientes deducciones para completar nuestra definición del problema:

- Nombrado de las notas con n seguido de un número según el orden de aparición. En MusicXML las notas se colocan de manera ordenada.
- Cálculo de la posición de la nota dentro del compás a partir de la duración de las notas anteriores de ese mismo compás.
- Cálculo de la duración de las notas a partir de divisions y duration. La operación que se realiza es la siguiente:

  *(16/divisions) * duration*

El valor de una negra es 16. De ahí el 16 por el que se divide.

Por otro lado se hacen los siguientes añadidos:

- Asignación a la nota de velocidad 0. *(= (speed n1) 0)*
- Nota *start* y nota *finish*
- Predicados para la secuencia
- Añadido de la meta.

La situación final que se obtendría sería la correspondencia de la siguiente figura:

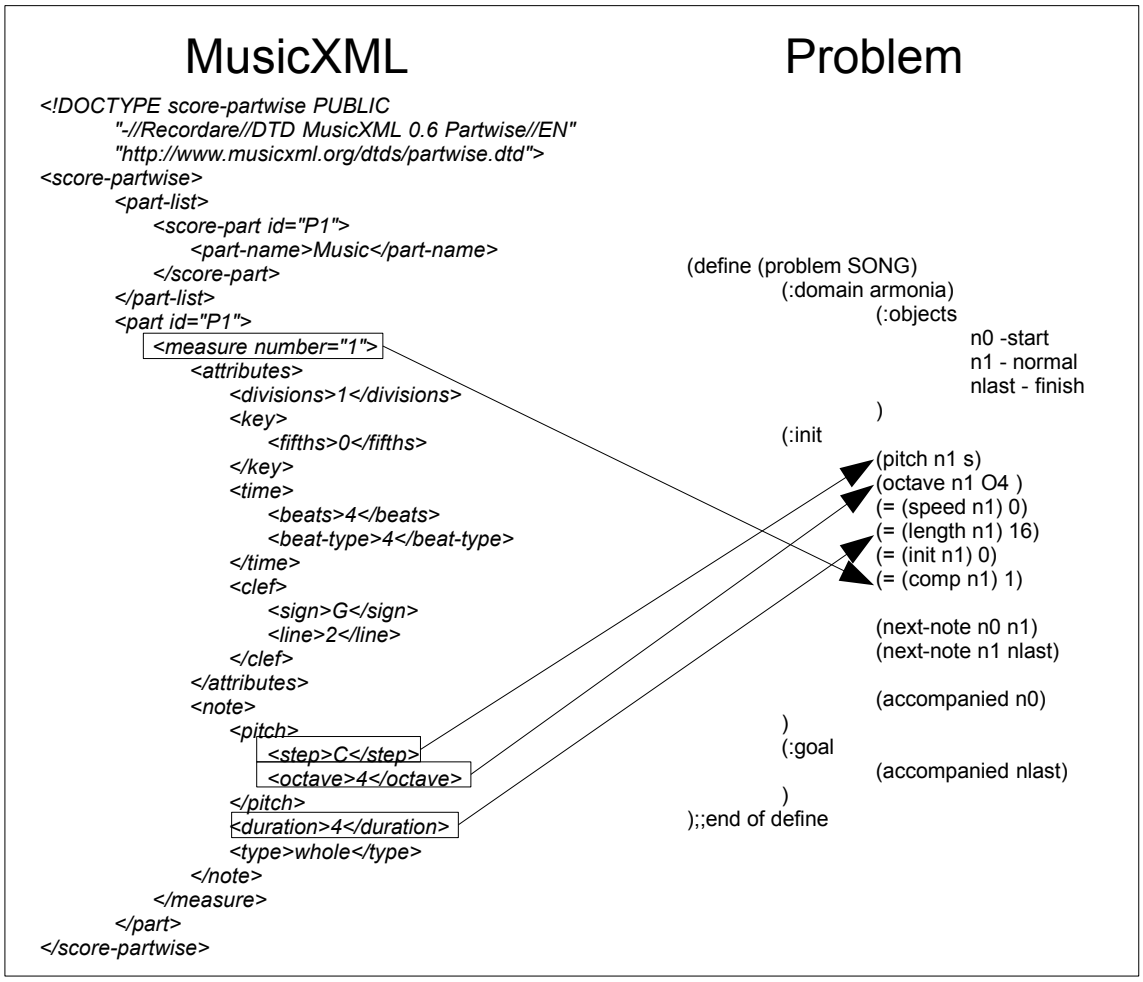


Ilustración 8. Correspondencia MusicXML - Problema.

4.2.3 Correspondencia Base de Datos Dominio

En esta sección se describe cómo se transforman los datos que se encuentran en la base de datos a los ficheros necesarios para la obtención de la salida total.

En primer lugar se explica el contenido de la base de datos. Después cómo se obtiene el fichero de dominio a partir de esta base de datos y por último el fichero de acordes.

Todos estos ficheros son imprescindibles para la obtención de una salida completa.

4.2.3.1 Base de Datos

La información contenida en la base de datos es la referente a los acordes. Un acorde es un conjunto de notas que suenan simultáneamente. Un acorde acompaña a una nota.

Fuera de nuestro dominio, una nota puede estar acompañada por varios acordes y un acorde puede acompañar a varias notas. Dadas las restricciones de nuestro dominio: monofonía en la melodía y un solo acompañamiento, la base de datos tiene una estructura acorde a estas restricciones.

La estructura, sin embargo, no es restrictiva y no cierra el camino a nuevos modelos de reglas.

Para mantener un orden en los acordes, estos se pueden agrupar por estilos. Los estilos no tienen por que ser un reflejo de un estilo musical obligatoriamente, pueden ser también una forma de denotar un grupo de acordes con características comunes. Por ejemplo, si se quisiese hacer un grupo de acordes que además de la tríada principal contuviesen la quinta dos octavas por debajo, se podría agrupar estos acordes dentro de un estilo.

Un estilo se identifica por un *id* y tiene un *nombre* y una *descripción*.

Un acorde pertenece a un estilo. Está también identificado por un id y además tiene un nombre y una descripción. Tanto el *id* del estilo como el *id* del acorde son claves autonumeradas generadas por la base de datos.

A la hora de relacionar los acordes con las notas como se comentaba al principio de esta sección, se necesita saber a qué notas responder y con qué notas. Esta información es dada por dos tablas: *action* e *inter*.

Action es una tabla que indica ante qué tonos está permitido poner un acorde. Está identificada por el acorde y el tono ante el cual responde.

Inter dicta qué notas forman el acorde. Habrá una entrada por cada nota del acorde y están identificadas por el id del acorde, la altura de la nota y la octava a la que pertenecen.

El siguiente esquema Entidad/Relación muestra la estructura de la base de datos:

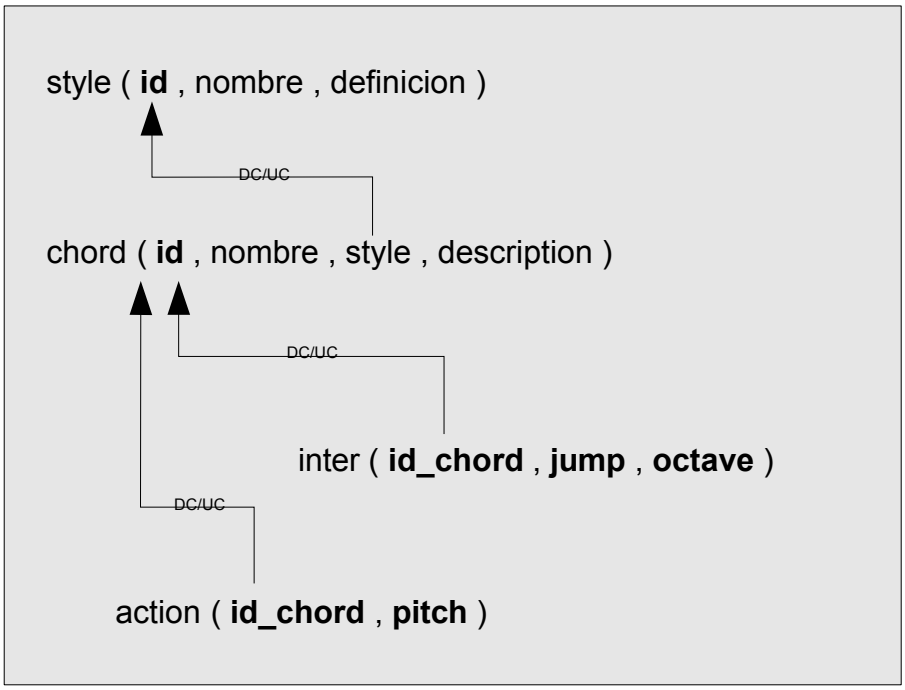


Ilustración 9. Esquema Entidad Relación de la base de datos de MusicMaker.

La siguiente tabla contiene las referencias entre los nombres de los campos de la base de datos y una definición de tales.

Tabla	Campo	Definición	Referencia
chord	**id**	número identificador del acorde	
	nombre	nombre del acorde	
	style	estilo al que pertenece el acorde	**style.id**
	description	descripción del acorde	
style	**id**	identificador del estilo	
	nombre	nombre del estilo	
	frdefinición	definición sobre el estilo	
action	**id_chord**	identificador del acorde al que pertenece	**chord.id**
	pitch	tono que dispara el acorde identificado	
inter	**id_chord**	identificador del acorde al que pertenece	**chord.id**
	jump	tono que se dispara por el acorde	
	octave	octava del tono que se dispara	

Tabla 37. Definición de los campos de la base de datos.

En cuanto al tipo de campos que se están utilizando la siguiente tabla muestra la información para cada uno de los campos

Tabla	Campo	Tipo de Campo
chord	**id**	Entero de longitud 11
	nombre	Cadena de caracteres de longitud 25
	style	Entero de longitud 11
	description	Cadena de caracteres de longitud 250
style	**id**	Entero de longitud 11
	nombre	Cadena de caracteres de longitud 10
	definicion	Cadena de caracteres de longitud 250
action	**id_chord**	Entero de longitud 11
	pitch	Cadena de caracteres de longitud 5
inter	**id_chord**	Entero de longitud 11
	jump	Cadena de caracteres de longitud 5
	octave	Cadena de caracteres de longitud 5

Tabla 38. Tipado de los campos de la base de datos.

4.2.3.2 Creación del Dominio

En la definición del dominio ya se explicó cuál era el cometido de cada una de las partes. Esta

sección simplemente explica cuál es la relación entre cada uno de los campos de la base de datos referente a un acorde con su homólogo en el dominio.

También se trata la forma de conseguir la creación en términos de pseudocódigo.

La información referente al fichero de dominio se encuentra en dos tablas de la base de datos:

- La tabla chord: para obtener el nombre del acorde y el id.
- La tabla action para obtener las precondiciones de cada una de las acciones del acorde.

Como el nombre de la segunda tabla indica, lo que se debe obtener al final son las acciones. Estas acciones junto con los contenidos que se explicaron en la sección [4.2.1.1] generan completamente el dominio.

En la sección de constantes se han de indicar todos los acordes que se vayan a utilizar. Luego de cada acorde, se toman las acciones y se indican. El pseudocódigo siguiente generaría todos los acordes:

```
Chords = query("Select id,nombre from chord;")
for Chords as chord
{
   Actions = query("select pitch from inter where id_chord = " + chord.id
" ;")
   contador = 0
   for Actions as action
   {
      print   (:action <chord.nombre>_<contador>
      print   :  parameters (?n1 ?n2 - note)
      print    : precondition
      print         (and
      print            (pitch ?n1 <action.pitch>)
      print            (and
      print               (accompanied ?n2)
      print               (next-note ?n2 ?n1)
      print            )
      print         )
      print      :effect
      print         (and
      print            (acomp ?n1 <chord.nombre>)
      print            (accompanied ?n1)
      print         )
      print   )
      contador ++
   }
}
```

Código 19. Pseudocógido de generación de acordes.

4.2.3.3 Creación del fichero de acordes

Cuando se vaya a generar una melodía a partir del plan obtenido, se ha de saber interpretar la salida. Saber qué tonos forman qué acorde. Se podrían hacer consultas a la base de datos en el momento de generar la salida. Sin embargo esto anularía la posibilidad de generar los datos desde otra fuente. Por motivos de reutilización se debe tener también el contenido de los acordes en fichero.

El formato de este fichero es muy simple. Consta del nombre del acorde seguido de las notas que lo forman. Para distinguir el principio de un acorde se coloca un punto antes de su nombre.

La información acerca de las notas que forman los acordes se encuentra en las tablas chord e inter. El siguiente Pseudocódigo generaría el fichero de acordes para todos los acordes:

```
Chords = query("Select id,nombre from chord;")
for Chords as chord
{
  print .<chord.nombre>
  Notas = query("select jump,octave from inter where id_chord = " +
chord.id + ";")
  for Notas as nota
  {
    print <nota.jump>,<nota.octave>
  }
}
```

Código 20. Pseudocódigo de generación del fichero de acordes.

4.2.4 Obtención del MIDI salida

Una vez que se ha conseguido un plan (acompañamiento) para el problema que teníamos (melodía), es normal que se desee escuchar el resultado. Siendo la naturaleza del proyecto musical, y MIDI un formato digital común en la informática; se ha elegido este como formato de los ficheros de salida.

La conversión de la melodía y el acompañamiento a MIDI se realiza en dos fases. Obligatoriamente porque la melodía y el acompañamiento están en dos sitios distintos.

- La melodía está en el problema con formato de problema PDDL [5.4]
- El acompañamiento está en el plan con el formato descrito en [5.5]

Se van a poder producir entonces varios tipos de salidas:

- Melodía únicamente.
- Melodía + acompañamiento.
- Acompañamiento únicamente.

La especificación que se realiza en el problema de cada nota es bastante similar a los datos que constituyen la especificación MIDI. MIDI soporta mucha más información, sin embargo, así que existen un montón de valores que permanecerán con valores por defecto.

4.2.4.1 Transformación de valores propios de MIDI

Algo a tener en cuenta cuando se intenta obtener salida en otro formato es cómo difieren los valores. En este caso se observan varias diferencias.

Aunque la información es casi la misma, entre nuestro dominio y MIDI, los valores no son equivalentes. Esto es debido a que a la hora de desarrollar el dominio, éste fue basado en MIDI, pero no enfocado hacia la transformación, sino hacia la comprensión humana y la facilidad de utilización dentro de la planificación.

En primer lugar se debe especificar cómo traducir nuestro pitch + octave en el valor numérico aceptado por MIDI. La siguiente tabla muestra la correlación[1]:

1 Se supone que la tónica de salida es Do.

Octave	Pitch	MIDI
1	P1	24
1	P1s	25
1	P2	26
1	P2s	27
1	P3	28
1	P4	29
1	P4s	30
1	P5	31
1	P5s	32
1	P6	33
1	P6s	34
1	P7	35
2	P1 (P1/O1 + 1Octava)	36 (= 24+12)
3	P1	48
4	P1	60
5	P1	72
6	P1	84
7	P1	86
8	P1	98

Tabla 39. Relación Pitch/Octave MusicMaker y MIDI.

En segundo lugar se debe transformar nuestra numeración de las duraciones a MIDI. Ya que MIDI no entiende el concepto de nota, lo que habrá que calcular son los momentos en los que la nota empieza a sonar y deja de hacerlo.

La aplicación completa del tiempo significa también el cálculo del compás. Ya que MIDI no contiene el concepto de compás en su especificación, se deberá utilizar nuestra numeración de compás, el valor de los tempos y el valor de la duración de las notas MIDI para añadírselo al valor de iniciación de cada nota.

De igual manera, para calcular la finalización de la nota habrá que basarse en el valor antes calculado y en la duración de la nota y sumarlos para obtener el momento de finalización de la nota.

El cálculo del momento de ambos eventos será por tanto una función que sume la duración del número de compases anteriores, las notas anteriores del mismo compás y en caso del evento 'off' la duración de la nota en sí.

En definitiva la transformación de una nota de nuestro formato en MIDI se podría especificar en pseudocódigo de la siguiente manera:

```
pitch = note.pitch
octave = note.octave
speed = note.speed
length = note.length
init = note.init
comp = note.comp

//get correct values for the note pitch
notePitch = calculate(pitch, octave)

//get correct values for the message moments:
messageOnTime = init + (comp * 4 * 16)
messageOffTime = init + length + (comp * 4 * 16)

//create the messages
messageOn = createMIDIMessage ('ON', notePitch, speed, messageOnTime);
messageOff = createMIDIMessage('OFF', notePitch, speed, messageOffTime);
```

Código 21. Pseudocódigo de generación de una nota en MIDI.

Hay que tener en cuenta, que en función de la implementación específica, puede que debamos identificar los silencios, no como notas sino como ausencia de estas. En el código final, puede que deban ser ignoradas, pero ya que cada nota está completamente definida por sus atributos y no depende de otras, el código es robusto.

4.2.4.2 Transformación del problema a MIDI

La transformación del problema (la melodía) en un fichero MIDI consiste en recorrer la secuencia de notas e ir transformándolas en eventos tal y como se describieron en la sección [4.1.1.2.2.3.2].

Únicamente puede que surja algún tipo de inconveniencia por la definición errónea de la nota tónica. Esta debe ser la misma para los acordes si se quieren conjuntar.

Para simplificar la extracción de las notas, en la definición del dominio se comentaba que las notas deben ser especificadas en orden. Esto no es una restricción del lenguaje PDDL, ya que en la descripción del estado inicial esto no importa, sin embargo nos facilita el proceso de corrección, comprensión y extracción.

4.2.4.3 Transformación del fichero de acordes a MIDI

El plan (fichero de acompañamiento) como ya se explicó con anterioridad no tiene valía musical por sí solo. Se necesita del fichero de acordes para que pueda ser interpretado. El fichero de acordes contiene las notas que forman cada acorde.

Para poder acompañar una nota, se obtienen las propiedades de la nota que acompañamos y se aplican esas propiedades a las notas del acorde. Las propiedades copiadas, obviamente, no tienen que ver con la altura de las notas que forman el acorde, pero estas alturas tienen que ser también calculadas a partir de la nota tónica. El proceso es el mismo que en [4.2.4.1] para cada una de las notas del acorde.

5 Manual Técnico

En esta sección se expondrá toda la teoría de conversión y ejecución expuesta en el apartado anterior desde el punto de vista de la implementación concreta. Es decir, se determina cómo se deberían hacer las cosas con la programación específica de cada una de las partes.

Es, por tanto, lógico, que se hable en esta parte del código Java, concreto, de las librerías sobre las que se apoya, del planificador en concreto que hemos utilizado y de los problemas específicos de la implementación que en el planteamiento de las secciones anteriores no se podía prever.

El primer punto es una descripción de las clases y paquetes más importantes de la aplicación.

Para continuar, se trata la implementación del creador del dominio. A partir de la base de datos y de la selección de acordes que se disponga. A partir de estos elementos también, en el siguiente punto, se relata la codificación del código que genera la codificación de acordes.

El código que crea el problema es la siguiente sección. En éste punto es dónde se habla de la importación desde el formato MusicXML, porque es el único contexto en el que esto tiene sentido.

La forma de llamar a la ejecución al planificador es también importante. Esto es explicado a continuación.

Por último, y muy importante para las pruebas y la valoración de las salidas, la exportación a MIDI. En esta sección se trata cómo cada una de las partes específicamente es transformada en el estándar con ayuda de bibliotecas propias de Java que simplifican mucho la tarea.

5.1 Paquetes y clases

El código está dividido en tres paquetes. La separación está llevada a cabo en función de la funcionalidad y los recursos que utilizan. Así los paquetes que lidian con la base de datos y la interfaz gráfica están incluidos en el paquete gui. El paquete noting incluye la abstracción de los elementos musicales y, por último, el paquete MusicXML contiene el elemento de transformación de XML a PDDL.

5.1.1 Paquete noting

El paquete noting contiene una serie de clases las cuales hacen todas referencia a información musical. El siguiente diagrama de clases representa el paquete noting:

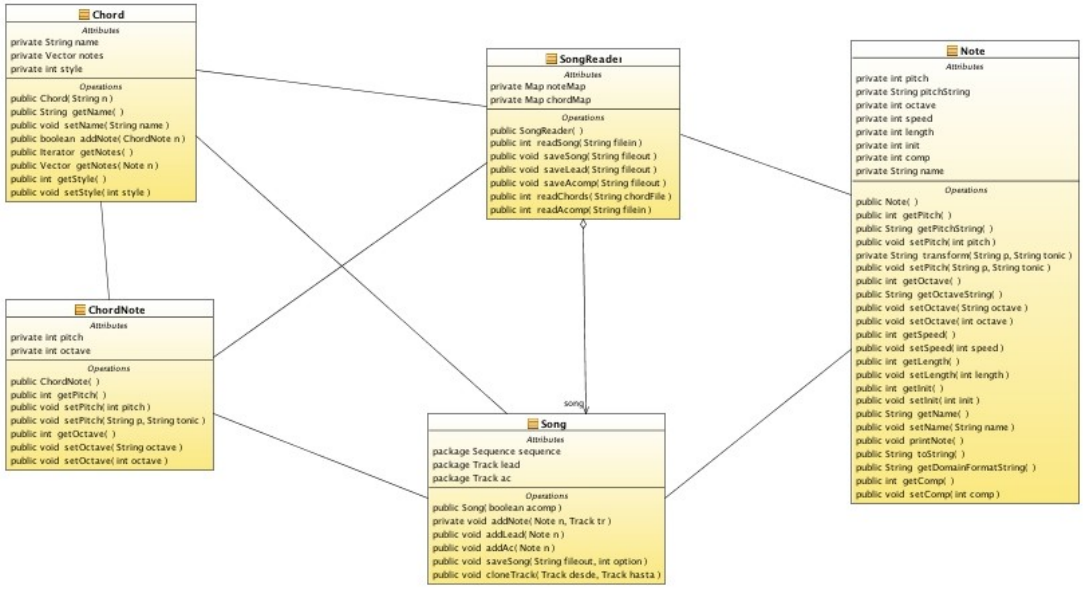


Ilustración 10. Diagrama de clases del paquete noting.

- Note.java

La clase Note representa una nota de nuestro dominio. Esto es sin embargo un concepto abstracto, pues la nota en el dominio no existe más que como la reunión de ciertos enunciados. La clase, por tanto, contiene estos enunciados como atributos. Cada atributo tiene sus métodos set y get. Los distintos atributos que se encuentran son los siguientes:

- pitch: representación numérica de la altura de la nota.
- PitchString: representación textual de la altura de la nota.
- Octave: Octava de la nota.
- Speed: velocidad de reproducción MIDI de la nota.
- Length: duración de la nota.
- Init: posición dentro del compás que ocupa la nota.
- Comp: compás dentro de la pieza donde está la nota.
- Name: nombre de la nota.

Cada uno se corresponde también con un elemento de MusicXML. Se utiliza la misma abstracción para pasar de MusicXML al problema y del problema a MIDI. Se profundiza en la generación de MIDI en la sección [5.6], y en la extracción de MusicXML en la sección [4.2.2].

Además de los métodos set y get de cada atributo, existe un método que se encarga de dar la salida para escribir en el problema. El código del método es el siguiente y es bastante auto explicativo. Como se explicaba antes, cada atributo de la clase se corresponde con un enunciado. Este método imprime todos los enunciados para los atributos de una nota:

```
public String getDomainFormatString()
{
   String ret = "(pitch " + this.getName() + " " +
      this.getPitchString() + ")\n"+
      "(octave " + this.getName() + " " + this.getOctaveString() +  " )\n"+
      "(= (speed " + this.getName() + ") " + this.getSpeed() + ")\n"+
      "(= (length " + this.getName() + ") " + this.getLength() + ")\n"+
      "(= (init " + this.getName() + ") " + this.getInit() + ")\n"+
      "(= (comp " + this.getName() + ") " + this.getComp() + ")\n";
   return ret;
}
```

Código 22. Método Note.getDomainFormatString().

- ChordNote.java

La clase ChordNote es una simplificación de la clase Note. No necesita en un principio todos sus atributos, pues dada la manera de funcionar del sistema los heredará más tarde de una nota. Los objetos de esta clase van a formar parte de un acorde, clase Chord. Tan solo necesitan tener un pitch y un octave.

- Chord.java

La clase Chord representa un acorde. Un acorde, como elemento abstracto que se consideró en partes anteriores, tiene un nombre, pertenece a un estilo y además está formado por una serie de notas.

Esos son sus tres atributos.

- Name: String que identifica al acorde.
- Style: Valor entero para identificar su estilo.
- Notes: Vector que contiene objetos de la clase ChordNote que son las notas que forman el acorde.

Como se ha mencionado en la clase anterior, las notas de un acorde heredan sus atributos de la nota a la que acompañan. Cuando un acorde finalmente acompaña a una nota, no se concreta el acorde en sí, sino que se concretan sus notas. De cada uno de los objetos ChordNote, se crea un objeto Note, que tiene los atributos pitch/octave de ChordNote y el resto de la nota a la que acompaña. De esto se encarga el método getNotes(Note n) de la clase Chord. El código de este método es el siguiente:

```
public Vector getNotes(Note n)
{
   Vector notas = new Vector();
   Iterator it = notes.iterator();
   while (it.hasNext())
   {
      Note newnote = new Note();
      ChordNote cn = (ChordNote)it.next();
      newnote.setComp(n.getComp());
      newnote.setInit(n.getInit());
      newnote.setSpeed(n.getSpeed());
      newnote.setOctave(cn.getOctave());
      newnote.setPitch(cn.getPitch());
      newnote.setLength(n.getLength());
      notas.add(newnote);
   }
   return notas;
}
```

Código 23. Método Chord.geNotes(Note n).

- Song.java

Los objetos de esta clase se encargan de generar la salida en formato MIDI. Utilizando las bibliotecas de java para el manejo de MIDI que ya se mencionaron [4.1.4.1], se definen aquí todas las propiedades del MIDI necesarias para que la salida tenga sentido. Por otro lado se pueden añadir a la canción notas como objetos Note y son transformadas a eventos MIDI. Este proceso se define más en profundidad en Exportación MIDI [5.6].

- SongReader.java

Esta clase lee los ficheros que se han generado y los convierte todos en notas que envía a un objeto de la clase Song para que sean exportados a MIDI. La lectura de los datos se realiza de tres fuentes:

- El fichero de acordes: Se lee la estructura de acordes.
- El fichero de canción: Se leen las notas de la canción.
- El fichero de acompañamiento: Se lee qué acorde acompaña a cada nota y se genera el acorde (en el fichero acordes se indica de manera relativa, aquí se transforma a absoluta) y se añaden las notas a Song.

5.1.2 Paquete xmlmusic

Este paquete tiene una sola clase MusicParser. La clase está definida como una extensión del DefaultHandler de la biblioteca org.xml.sax. DefaultHandler está dentro del paquete helpers y ayuda a crear una clase que contiene métodos los cuáles responden a distintos eventos cuando se lee el xml.

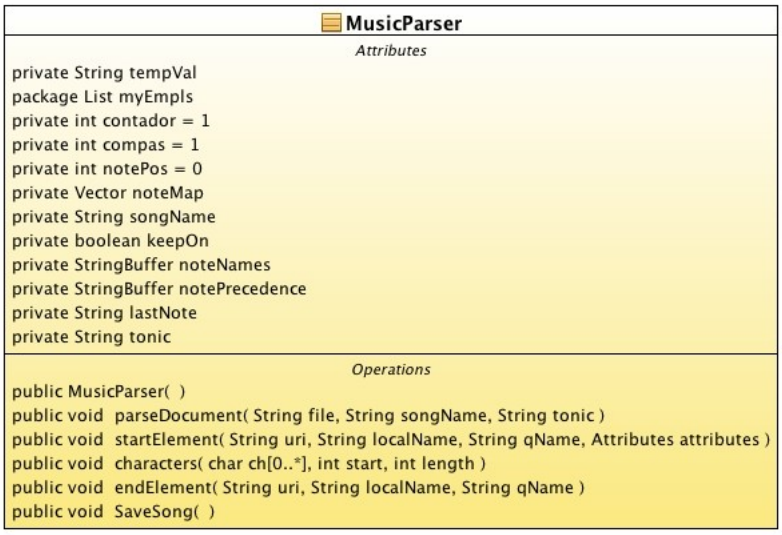


Ilustración 11. Diagrama de clases del paquete xmlmusic.

En la sección [4.2.2] se vio que eran muy pocos los elementos del xml que se correspondían directamente con un elemento de nuestras definiciones de dominio.

Cada una de las correspondencias entre estos elementos son detectados con los eventos startElement y endElement. Estos eventos son disparados de manera automática en el parseo y se encarga de ello el framework xml de java que ya se mencionó.

5.1.3 Paquete gui

El paquete gui contiene la clase principal de la interfaz de usuario: MusicMaker. No aparece en el esquema pues sus métodos son irrelevantes y creados por el editor de NetBeans [NBe]. De igual manera han sido creadas automáticamente las clases Chord, Style, Inter, InterPK, Action y ActionPK; éstas para la gestión automática de la persistencia sobre base de datos de las modificaciones que realizamos sobre la base de datos.

Las clases ChordFileBuilder y DomainBuilder sí han sido construidas específicamente para la exportación a PDDL.

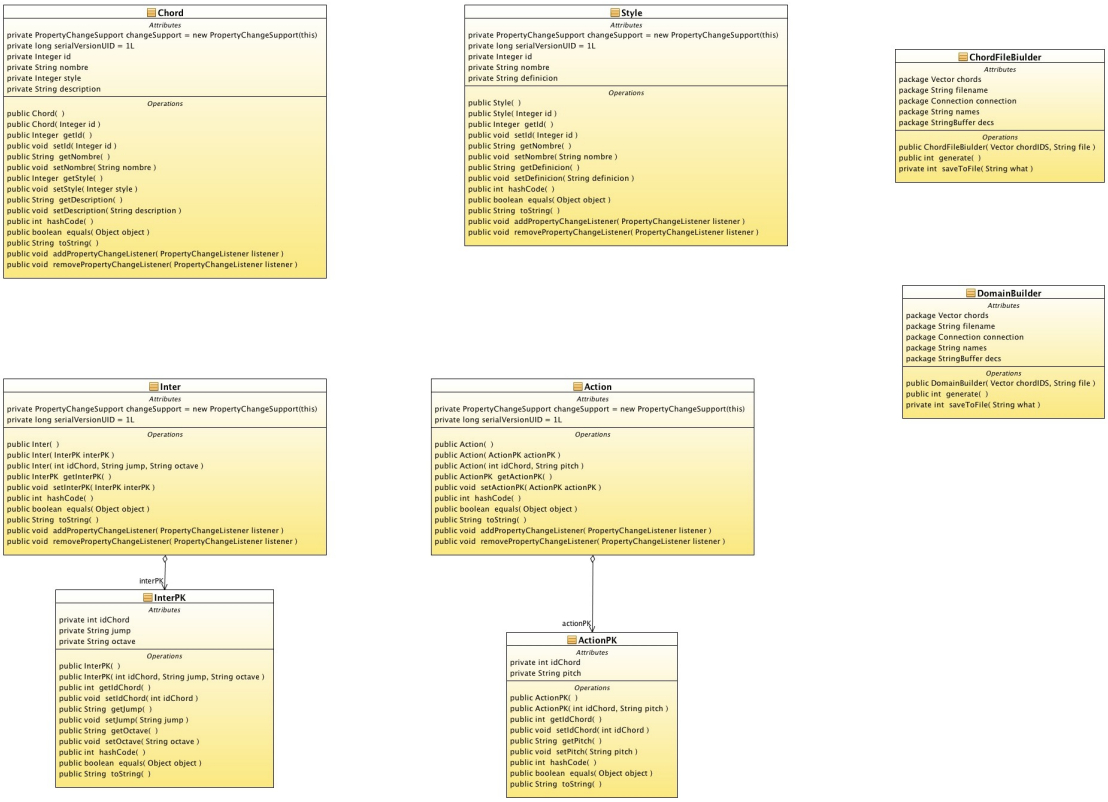


Ilustración 12. Diagrama de clases del paquete gui.

5.2 Creación del dominio

La clase encargada de crear el dominio es DomainBuilder. Es una clase simple del paquete “gui”. Su constructor recibe como parámetros un vector que contiene como Strings los ids de los acordes que se quieran incluir en el dominio y el nombre del fichero donde se quiere salvar la salida.

```
/** Constructs a DomainBuilder Object with the given chords as the chords
   * to be included in the domain and as output file the given filename.
   *
   * @param chordIDS Vector of Strings of the ids of the
   * chords that are desired to be included in the domain
   * @param file String filename where we want the domain to be generated
*/
public DomainBuilder(Vector chordIDS, String file)
```

Código 24. Constructor DomainBuilder.

La clase DomainBuilder tiene sólo un método público. Éste es generate(). El método realiza el pseudocódigo que se especificaba en la creación del dominio para la especificación de las acciones de cada uno de los acordes.

Además de ello, tiene que guardar un fichero de dominio bien formateado para PDDL. Es por ello que añade todos los datos que se veían en la descripción de este formato.

```
/** This method generates the domain and saves it to file
   *
   * @return -1 on failure, 0 on success
*/
public int generate()
```

Código 25. Método DomainBuilder.generate().

Para generar el texto se ha utilizado un StringBuffer que permite añadir texto de forma que él solo gestiona el aumento de tamaño automáticamente. Para el acceso a la base de datos como ya se dijo se ha utilizado el conector Java JDBC y las sentencias son compatibles con SQL aunque la base de datos es MySQL.

Lo último que realiza el método generate es la llamada a un método privado de la clase que se encarga de escribir en fichero. Para ello utiliza las habituales librerías del sdk de java "java.io". El método se llama saveToFile.

```
/** This method saves an string to file. The filename is the one of the
class
   *
   * @param what string to save to file
   * @return -1 on failure, 0 on success
*/
private int saveToFile(String what)
```

Código 26. Método DomainBuilder.saveToFile(String what).

5.3 Creación del fichero de acordes

De nuevo, como en el creador de dominios se dispone de una clase creadora de ficheros de acordes: ChordFileBuilder. Su constructor es idéntico al del dominio:

```
/** Constructs a ChordFileBuilder object with a set of chord ids and
* a filename defined
*
* @param chordIDS the set of chord ids which are desired to be saved
* @param file filename where the chord file will be saved
*/
public ChordFileBuilder(Vector chordIDS, String file)
```

Código 27. Constructor ChordFileBuilder.

Al igual que en el apartado anterior tenemos los métodos generate y saveToFile:

```
/**Generates de chord File and saves it.
*
* @return -1 on failure, 0 on success
*/
public int generate()
```

Código 28. Método ChordFileBuilder.generate().

```
/** This method saves an string to file. The filename is the one of the
class
*
* @param what string to save to file
* @return -1 on failure, 0 on success
*/
private int saveToFile(String what)
```

Código 29. Método ChordFileBuilder.saveToFile(String what).

La forma de proceder es exactamente la misma que en el caso anterior. Las particularidades de acceso a la base de datos y escritura en disco de Java son también iguales.

5.4 Creación del problema

En este punto se trata la importación desde el formato MusicXML al formato específico de nuestro dominio PDDL.

Esto abarca varios aspectos. En primer lugar se debe explicar el parseo de xml. En segundo lugar explicar una serie de clases que harán referencia a nuestras notas, acordes y demás información. Por último lugar explicar cuál es el proceso por el cual se guarda una descripción de problema en estas clases en el fichero de salida del problema.

Se deben capturar eventos en dos lugares según van surgiendo en el xml. Primeramente la apertura de un tag xml:

```
public void startElement(String uri, String localName, String qName,
Attributes attributes) throws SAXException {
        if (keepOn)
        {
            //reset
            tempVal = "";
            if (qName.equalsIgnoreCase("note")) {
                //create a new instance
                nota = new Note();
                nota.setName("n"+contador++);
            }
        }
    }
```

Código 30. Método Musicparser.startElement (…).

Como se puede observar, sólo capturamos la apertura de una nota. Ésta apertura indica que se tiene que crear un objeto Note nuevo en el cual se almacenan los próximos tags que vayan llegando. La razón por la que no se actúa de ninguna manera en la apertura del resto, aunque parezca lógico que se inicializasen esos datos, es porque cuando se abre el tag no sabemos qué habrá dentro, esta información está disponible a la hora de cerrar cada tag:

```
public void endElement(String uri, String localName, String qName) throws
SAXException {
        if (keepOn)
        {
            if (qName.equalsIgnoreCase("note")) {
                nota.setComp(compas);
                //add it to the list
                noteMap.add(nota);
                noteNames.append(nota.getName() + " ");
                if (lastNote != null)
                    notePrecedence.append("(next-note " +
                                          lastNote + " " +
                                          nota.getName() + ")\n");
                lastNote = nota.getName();
                System.out.println(nota.toString());

            } else if (qName.equalsIgnoreCase("rest")) {
                nota.setPitch("S", this.tonic);
                nota.setOctave(4);

            } else if (qName.equalsIgnoreCase("step")) {
                nota.setPitch(tempVal, this.tonic);

            } else if (qName.equalsIgnoreCase("duration")) {
                int thisLength = (Integer.parseInt(tempVal)*8);
                nota.setLength(thisLength);
                nota.setInit(notePos);
                notePos+=thisLength;
            } else if (qName.equalsIgnoreCase("measure")) {
                compas++;
                notePos=0;
            } else if (qName.equalsIgnoreCase("octave")) {
                nota.setOctave(Integer.parseInt(tempVal));

            } else if (qName.equalsIgnoreCase("part")) {
                notePrecedence.append("(next-note " +
                                          lastNote + " nlast)\n");
                keepOn = false;
            }
        }
    }
```

Código 31. Método Musicparser.endElement(String uri, String localName, String qName).

Nótese que en el cierre de cada tag se ponen a la nota actual sus valores (step, duration, measure, octave y part). La nota porque cada vez que se parsea el comienzo del tag note se crea una nueva.

Después de haber terminado con el fichero se dispondrá de toda la información guardada en tres variables: noteMap, noteNames y notePrecedence. Sólo faltará guardarlo en fichero. En el código ejemplo de problema [15] se observan las partes, se deberá ahora ampliar esa información para todas las notas, así cada una de las secciones se completa y el resto permanece intacto:

```
public void SaveSong()
    {
        StringBuffer linea = new StringBuffer();

        linea.append("(define (problem SONG)\n");
        linea.append("\n");
        linea.append("\t(:domain armonia)\n");
        linea.append("\n");
        linea.append("\t\t(:objects\n");
        linea.append("\t\t\tn0 -start\n");
       //imprimimos todos los nombres de las notas
        linea.append("\t\t\t" + noteNames);
        linea.append(" - normal\n");
        linea.append("\t\t\tnlast - finish\n");
        linea.append("\t\t)\n");
        linea.append("\t(:init\n");
        linea.append("\n");
        Iterator i = noteMap.iterator();
        while (i.hasNext())
            linea.append((Note)i.next()).getDomainFormatString()+"\n");
        linea.append("\n");
        linea.append(notePrecedence.toString());
        linea.append("\n");
        linea.append("\t\t(accompanied n0)\n");
        linea.append("\t)\n");
        linea.append("\n");
        linea.append("\t(:goal\n");
        linea.append("\t\t(accompanied nlast)\n");
        linea.append("\t)\n");
        linea.append("\n");
        linea.append(");;end of define\n");

        //we save to file
        try {
            // Create file
            FileWriter fstream = new FileWriter(songName);
            BufferedWriter out = new BufferedWriter(fstream);
            out.write(linea.toString());
            //Close the output stream
            out.close();
        } catch (Exception e) {//Catch exception if any
            System.err.println("Error: " + e.getMessage());
        }
    }
```

Código 32. Método Musicparser.saveElement().

Toda la información recaudada del fichero xml queda por tanto expresada en términos que nuestro planificador entenderá.

5.5 Ejecución del planificador

La versión del planificador LPG que se utiliza es la 1.0. Como formato de los ficheros de entrada soporta PDDL2.2. La llamada dentro del código no está integrada, hay que ejecutar el comando como un proceso externo y capturar sus salidas. Para ello se utilizan distintos elementos de Java que lo permiten.

La clase que permite ejecutar comandos / programas externos es Runtime. De hecho, concretamente, se debe llamar a Runtime.getRuntime().exec(<programa>) para ejecutar el comando. Éste crea un proceso hijo donde se realiza la ejecución. La salida de tal ejecución se hace por la salida estándar, así que para mostrar el resultado se debe sustituir. Se crea un buffer de lectura

y se le indica que lea la salida para capturarla:

```
InputStream is = child.getInputStream();
BufferedReader br = new BufferedReader (new InputStreamReader (is));
while ((line = br.readLine())!=null)
        this.PlanOutputTextArea.append(line + "\n");
```

Código 33. Captura de la Salida Estándar.

Formatos de entradas y salidas.

Opciones -o para operador

-f para el problema

-out para la salida

Opciones de ejecución.

-speed (se busca una solución rápida sin tener en cuenta su calidad)

-quality (permite refinar las soluciones si el planificador encuentra una demejor calidad)

-n <max number of desired solutions>

```
Process child = Runtime.getRuntime().exec("lpg-td/lpg-td-1.0 -o " +
                            this.domainFileNameTextField.getText() +
                            " -inst_with_contraddicting_objects -f " +
                            this.songFileNameTextField.getText() +
                            " -speed -out " +
                            this.planFileNameTextField.getText());
```

Código 34. Sentencia de ejecución del planificador.

En cuanto a los argumentos del planificador, los parámetros son los siguientes:

- La opción -o permite definir cuál es el fichero de dominio: this.domainFileNameTextField.
- La opción -f permite definir cuál es el fichero de problema: this.songFileNameTextField.
- La opción -out permite definir el fichero donde se guarda el plan obtenido: this.planFileNameTextField.

Como opciones de ejecución de la planificación se pueden especificar los siguientes:

- opción -speed
- opción -quality
- opción -n <número máximo de soluciones>

5.6 Exportación MIDI

Al explicar el paquete noting ya se introdujeron las dos clases que se encargan de la exportación a MIDI de las canciones. El punto más importante a destacar en estas dos clases es la conversión en eventos MIDI de cada nota.

El método addNote de la clase Song es el encargado en última instancia de realizar esto:

```
private void addNote(Note n, Track tr) {
   int lengthner = 1;

   //si hay un silencio no hay nota que poner
   if (n.getPitchString()!=null)
      if (n.getPitchString().equalsIgnoreCase("s"))
         return;

   ShortMessage on = new ShortMessage();
   ShortMessage off = new ShortMessage();
   try {
      on.setMessage(ShortMessage.NOTE_ON,0,n.getPitch(),60);
      off.setMessage(ShortMessage.NOTE_OFF,0,n.getPitch(),60);
   } catch (InvalidMIDIDataException ex) {
      ex.printStackTrace();
   }

   tr.add(new MIDIEvent(on,
           (n.getInit() + (n.getComp()*4*16))*
           lengthner));
   tr.add(new MIDIEvent(off,
         ((n.getInit()+n.getLength()) + (n.getComp()*4*16))
         * lengthner));
}
```

Código 35. Método Song.addNote(Note n, Track tr).

6 Evaluación

Esta sección se exponen los resultados obtenidos en las pruebas realizadas para probar el formato de dominio de planificación. Como ya se explico a lo largo del documento, el dominio se genera a partir del set de acordes que deseemos.

Los problemas de planificación son, como no, canciones. Tales canciones han sido importadas desde el formato MusicXML al formato del dominio y luego se han realizado un número de planificaciones sobre ellas para probar su consistencia.

Se ha prestado atención a las salidas a las soluciones, a su correctitud y a su existencia. No así a temas de rendimiento.

En primer lugar se explican cada uno de los dominios que se han probado. Han sido cuatro con un nivel de complejidad musical creciente.

El siguiente punto trata cada uno de los problemas atendiendo a su fuente, sus propiedades y en distintas subsecciones cada uno de los resultados de las distintas pruebas con los distintos dominios. En total son 4 dominios, con 3 problemas distintos y dos ejecuciones por cada uno, lo que suma un conjunto de 24 pruebas las comentadas.

6.1 Dominios

Dado que el marco del proyecto abarca tanto la música como la planificación, los dominios utilizados no podían centrarse simplemente en el primero de ellos. Es por ello que a partir de los dominios que podrían denominarse musicales (dado que parten de reglas de armonía), se ha definido un dominio en el cual los acordes no tienen conocimiento ninguno. Además este dominio nos sirve para contraponer al resto y decidir si las reglas musicales se pueden aplicar de la manera en que se hacen. Los siguientes dominios tienen conocimiento, es decir, tienen una serie de restricciones musicales.

Su complejidad va avanzando en número de acordes, formando en todos los casos dominios completos. Para que un dominio sea completo todas las notas del problema deben poder ser emparejadas con un acorde. Aún teniendo en cuenta que nuestro acorde silencio puede acompañar a cualquier nota (lo que constituiría la decisión de no acompañar el acorde), se han propuesto los dominios de manera que en ausencia del silencio también se pudiesen acompañar todas.

6.1.1 Sin Conocimiento

En este primer dominio todos los acordes de la escala pueden acompañar a cualquier nota. La siguiente tabla muestra la relación de acordes – notas que acompaña.

Acorde	Notas acompañadas
I	P1, p2, p3, p4, p5, p6, p7, s
II	P1, p2, p3, p4, p5, p6, p7, s
III	P1, p2, p3, p4, p5, p6, p7, s
IV	P1, p2, p3, p4, p5, p6, p7, s
V	P1, p2, p3, p4, p5, p6, p7, s
VI	P1, p2, p3, p4, p5, p6, p7, s

VII	P1, p2, p3, p4, p5, p6, p7, s
S	P1, p2, p3, p4, p5, p6, p7, s

Tabla 40. Acordes del dominio Sin Conocimiento.

6.1.2 Iniciación

Este dominio comienza a dar conocimiento musical a las acciones. La selección de este dominio tiene como base el cubrir la totalidad de las notas con el mínimo de acordes que tengan sentido musical. Esto ya se explicaba en el estado del arte, pero la siguiente tabla lo resume.

Acorde	Notas incluidas / acompañadas
I	P1, p3, p5
IV	P4, p6, p1
V	P5, p7, p2
S	P1, p2, p3, p4, p5, p6, p7, s

Tabla 41. Acordes del dominio Iniciación

A la vez la tabla anterior es la expresión de las notas que puede acompañar cada acorde.

6.1.3 Clásico

Este dominio está formado por todos los acordes de la escala natural: I, II, III, IV, V, VI y VII. Cada nota acompaña a las notas por las que está formado.

Acorde	Notas incluidas / acompañadas
I	P1, p3, p5
II	P2, p4, p6
III	P3, p5, p7
IV	P4, p6, p1
V	P5, p7, p2
VI	P6, p1, p3
VII	P7, p2, p4
S	P1, p2, p3, p4, p5, p6, p7, s

Tabla 42. Acordes del dominio Clásico.

6.1.4 Completo

Este dominio constituye una elaboración del anterior con inversiones en los acordes. Por cada acorde, y a partir de sus notas constituyentes, formamos dos inversiones utilizando la nota principal, su quinta y la nota principal una octava por encima (segunda forma de cada acorde) o la nota principal, su tercera y la nota principal una octava por encima (tercera forma del acorde). Cada acorde sigue acompañando a las notas por las que está formado como muestra la siguiente tabla.

Acorde	Notas incluidas	Notas que acompaña
I	P1, p3, p5	P1, p3, p5
I2	P1, p5, p1	P1, p5
I3	P1, p3, p1	P1, p3
II	P2, p4, p6	P2, p4, p6
II2	P2, p6, p2	P2, p6
II3	P2, p4, p2	P2, p4
III	P3, p5, p7	P3, p5, p7
III2	P3, p7, p3	P3, p7
III3	P3, p5, p3	P3, p5
IV	P4, p6, p1	P4, p6, p1
IV2	P4, p1, p4	P4, p1
IV3	P4, p6, p4	P4, p6
V	P5, p7, p2	P5, p7, p2
V2	P5, p2, p5	P5, p2
V3	P5, p3, p5	P5, p3
VI	P6, p1, p3	P6, p1, p3
VI2	P6, p3, p6	P6, p3
VI3	P6, p1, p6	P6, p1, p6
VII	P7, p2, p4	P7, p2, p4
VII2	P7, p4, p7	P7, p4
VII3	P7, p2, p7	P7, p4

Tabla 43. Acordes del dominio Completo.

6.2 Problemas

El origen de los problemas son canciones. Se ha prestado atención al contenido musical de éstas. Se han transformado una por una. En ninguna de ellas se utilizan formas musicales fuera de los conceptos teóricos de música que se explicaron en el apartado de teoría musical. Esto quiere decir que no existe ninguna nota alterada en ninguna de estas canciones. La posibilidad de abordar piezas con notas alteradas sólo estriba en la necesidad de reglas para ellas. No así los cambios de escala, para los cuales el dominio no está enfocado.

También la longitud de las piezas ha sido truncada. La razón es que mayor longitud no habría demostrado ningún concepto distinto. Habría sido circunstancial.

Los tres problemas para los cuales se han comentado las pruebas están comentados en profundidad a continuación. Para cada uno de ellos se ha comentado la percepción personal del autor en cada uno de los dominios. Datos como el número de notas que contiene el fichero de problema, su tónica, su origen, etc. también están expuestos.

6.2.1 La chispa adecuada

Los datos fundamentales de esta melodía son los siguientes. El formato de partida fue un MIDI con origen en Internet. De ahí se importó al formato MusicXML utilizando una herramienta comercial y esa fue la entrada para el conversor de xml a problemas descritos en PDDL.

El extracto que se ha elegido de la pieza completa es el introducción de guitarra. Ésta es sin duda una de las piezas más conocidas del grupo de rock y se ha utilizado para contrastar los resultados con el original al que se acostumbra a escuchar.

Origen;

Formato: MIDI.

Fuente: Internet.

Composición original; Héroes del silencio.

Extracto: introducción de guitarra de la composición.

Tónica: Fa

Número de notas: 99.

Compás: 4/4

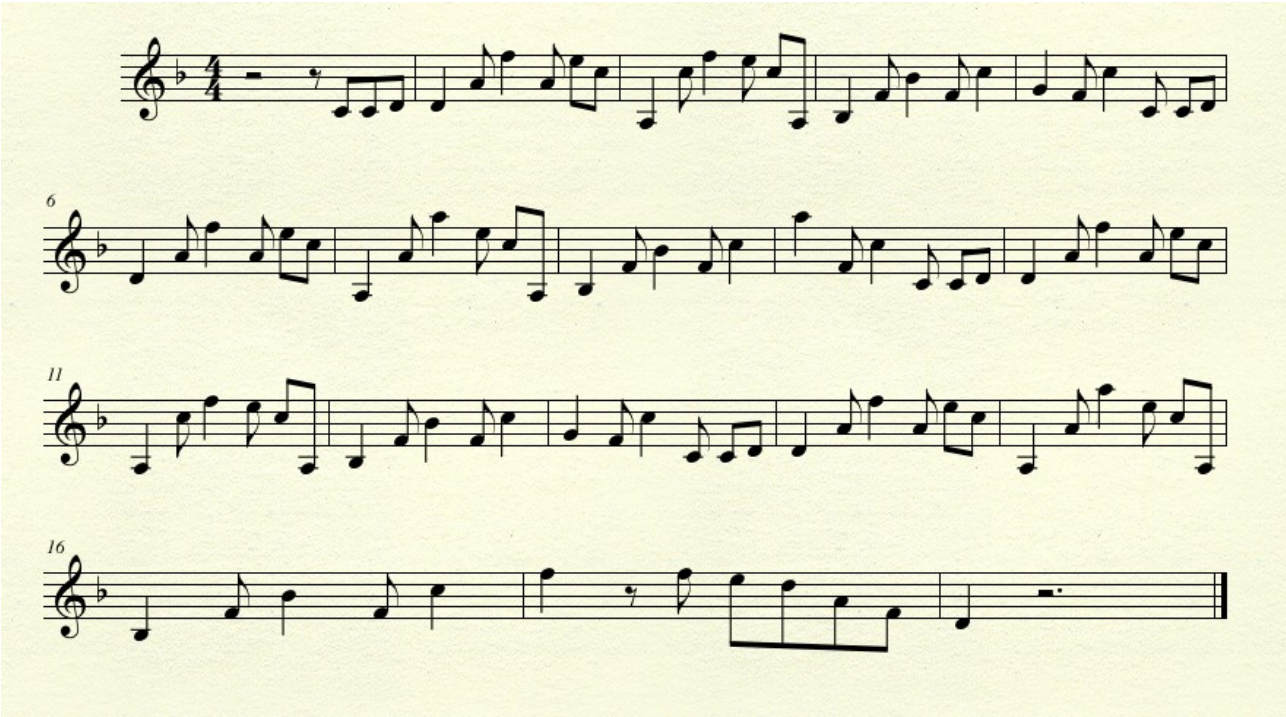


Ilustración 13 . Partitura de “La chispa adecuada”.

6.2.2 Werde munter, mein gemüte

Esta pieza de Johan Sebastian Bach es un pequeño coral que compuso y en el cual se basó para la composición “Jesus, joy of man's desiring”. Suele ser utilizada para la introducción al acompañamiento en el estudio de música.

La simplicidad de la melodía la hace perfecta para servir como estudio para nuestros dominios de planificación.

En este caso también se obtuvo en primer lugar el fichero en formato MIDI, se retocó y exportó a MusicXML con una herramienta comercial y se presentó en xml como entrada a MusicMaker.

Origen:

Formato; MIDI

Fuente: Internet (http://www.jsbchorales.net/jesu.shtml)

Composición original; Johan Sebastian Bach.

Extracto: piano mano derecha

Tónica: Do

Número de notas: 62

Compás: 4/4

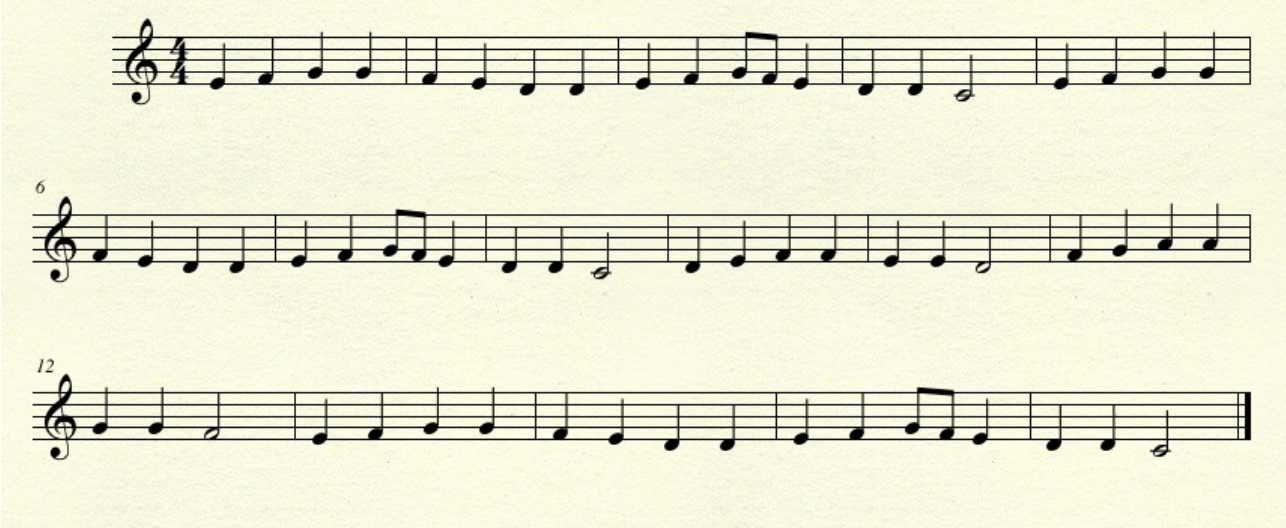


Ilustración 14 . Partitura de “Werde munter, mein gemüte”.

6.2.3 La cucaracha

Esta pieza popular ha sido escogida por conveniencia. Obtenida en formato MIDI de Internet y, como en los anteriores casos, retocada y exportada a MusicXML. El retoque supuso sencillamente la eliminación de las repeticiones de partes.

Origen:

Formato: MIDI

Fuente: Internet.

Composición original: Canción popular

Extracto: melodía principal reducida

Tónica: La

Número de notas: 167

Compás: 4/4

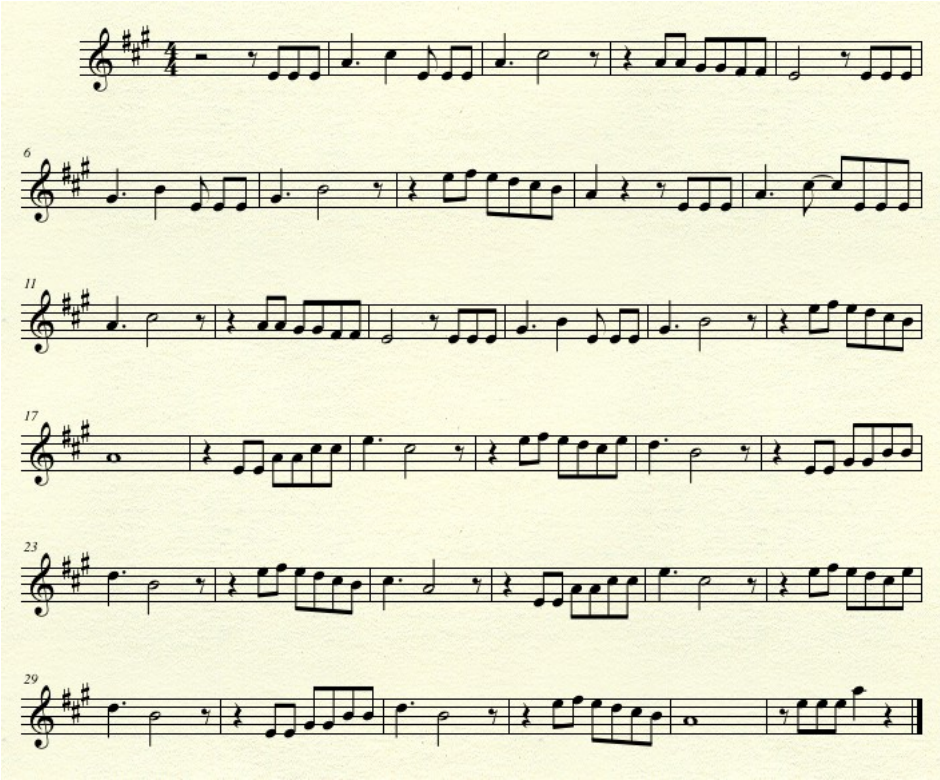


Ilustración 15 . Partitura de “La cucaracha”.

6.3 Resultados

Dado que como base para la realización de las pruebas se disponía de 4 dominios distintos y de 3 problemas, se optó por obtener dos planes distintos para cada una de las combinaciones dominio – problema. Eso supone un total de 24 salidas (4 dominio x 3 problemas x 2 ejecuciones).

Para apreciar cada uno de los dominios, se ha realizado una audición de todas las salidas para cada uno de ellos y a continuación se encuentran comentadas.

6.3.1 Sin Conocimiento

Los resultados de este dominio han sido los esperados. Las formas musicales no tienen ningún sentido en materia de armonía. Ocasionalmente se obtiene algún acorde correcto pero la naturaleza del dominio hace pensar que es aleatorio.

Por otra parte la inclusión del acorde silencio hace que se obtengan puntualmente patrones rítmicos curiosos. No guardan relación con la pieza en sí, pero tienen un cierto grado de musicalidad.

Los planes obtenidos, a grandes rasgos, tienen la forma esperada. Dado que el dominio no tiene información musical ninguna, la salida del planificador, a la vez, tampoco.

Tanto en el problema de la chispa adecuada como la cucaracha, se puede apreciar perfectamente la falta de conocimiento de la que hablamos. En la pieza de Bach, sin embargo, no es tan notable la disonancia.

6.3.2 Iniciación

Dado que este dominio ya contiene un grado de conocimiento musical, las planificaciones obtenidas tienen mayor sonoridad. Esto implica que, desde un enfoque musical, los acordes son correctos, pero, sin embargo, suenan monótonos.

El número de acordes es limitado y, a pesar de la poca longitud de los problemas, se distingue que siempre suenan los mismos. De hecho, únicamente el primer y quinto tonos tienen a posibilidad de ser acompañados por dos acordes distintos.

Se echa en falta la inclusión de reglas armónicas más complejas, que observen las notas como conjuntos y no como unidades separadas. A pesar de ello, la audición no es tan estridente como en el dominio anterior.

En la pieza de Bach, el sonido es sorprendentemente agradable. En la segunda prueba se finaliza con el grado I. Esto produce un gran efecto sobre la audición completa de la pieza. Por otra parte, las notas acompañadas con silencios dan ese ritmo característico del que hablábamos en el dominio sin conocimiento. Para el primer plan obtenido, el efecto de los silencios (mucho menos abundantes) llama la atención sobre notas específicas, lo cual aporta personalidad a la ejecución (incluso con las limitadas herramientas expresivas de que consta).

El problema de la cucaracha también tiene un sonido correcto. En ambos planes obtenidos, todos los acordes que se han colocado suenan de manera correcta. De igual manera todos los comentarios sobre monotonía y rítmica de los silencios se aplican en esta pieza también.

La chispa adecuada es una pieza más compleja, es por ello que los acordes que se han colocado son mucho menos agradables. Tal vez porque los intervalos son mucho más distantes y variados, y a la par la melodía tiene una estructura más enfocada al arpegio, fuerzan al oído a esperar un acorde determinado. Esto crea una sensación de tensión mayor y de ahí que parezca mucho menos apropiado el plan conseguido en ambas ejecuciones.

6.3.3 Clásico

Este dominio, como ampliación del anterior, contiene todos los acordes los cuales incluyen más repeticiones de las notas en ellos, lo cual se traduce en más posibles acompañamientos para cada nota. Sin embargo, a efectos prácticos, aún siendo la variedad de acordes mucho mayor, la mejoría no es muy notable.

En las seis audiciones se repiten los elementos del dominio de iniciación. Tanto el problema de Bach como el de la cucaracha tienen una sonoridad agradable. En cuanto a la mala sonoridad del problema de la chispa adecuada que nos encontrábamos con el dominio anterior, ahora no es notable y mejora mucho el acompañamiento con la mayor variedad de acordes.

Podemos, además, añadir, que igual que en el dominio sin conocimiento y el de iniciación, los silencios juegan un papel de caracterización. El ritmo y acentuación que consiguen en los planes es muy característico, es una cualidad que se potencia según crece la complejidad del dominio, pero también desaparece su asiduidad. Según aumenta el conjunto de acciones, los silencios van siendo cada vez menos frecuentes.

6.3.4 Completo

De igual manera que el dominio clásico se asemejaba al dominio de iniciación por ser su contenido muy parecido, podríamos esperar los mismo de los planes generados con este dominio. Sin embargo, la inclusión de las distintas inversiones de acordes aporta una riqueza que separa claramente este dominio del resto.

Los planes que se han generado contienen todas las buenas propiedades que hemos enunciado de los anteriores y algunas de las malas.

Los patrones rítmicos que se formaban en los escenarios anteriores ahora están mucho más potenciados por las inversiones. La aparición de acordes que no contienen tantas notas graves frente a otros que las contienen crean una sensación de énfasis en los segundos dotando al resultado final de una expresividad más rica.

La disminución de repeticiones en las formas de los acordes hace que las audiciones sean mucho más entretenidas y coloridas. De hecho los intervalos formados por las notas de los acordes de acompañamiento en ocasiones dan la sensación de ser lineas melódicas de bajo secundarias.

El problema de la melodía de Bach tiene en los dos planes obtenidos una sonoridad estupenda. El acompañamiento resulta natural y variado. Es agradable rítmico y completo. Se origina el fenómeno de la melodía secundaría en ambas piezas puntualmente que, sin embargo, no desvía la atención de la melodía principal haciendo que no sea reconocible. El resultado es muy satisfactorio y muy superior a los obtenidos con el resto de dominios.

Para el problema de la cucaracha, los planes no son tan ideales. En ocasiones el acompañamiento confunde la melodía al superponerse las octavas del acompañamiento y la melodía. Esto crea un efecto extraño y poco natural. Para esta pieza la riqueza en variedad de acordes que tan agradable era en la pieza anterior no resulta tan propicio. La caracterización rítmica no es tan patente dado que la pieza de por sí ya tiene un marcado acento originalmente. No obstante, el resultado no es desalentador, pues es un acompañamiento totalmente correcto.

La chispa adecuada obtiene unas planificaciones curiosas. Tiene una combinación de los dos casos anteriores. Mientras que por una parte ocurre el efecto de superposición de la cucaracha,no es tan sencillo confundir la linea melódica. En ocasiones el efecto del acompañamiento no suena muy natural, como ocurría con este mismo problema en otros dominios. Por otro lado tiene momentos de un acompañamiento perfecto. La complejidad mayor de la pieza puede ser como en casos anteriores un síntoma de esta causa. Sin embargo es en este dominio donde los planes para el problema son ampliamente superiores a los del resto.

6.4 Comentarios

Todas las pruebas poseen un paralelismo tanto en función del dominio como del problema. Si nos fijamos en función del problema observamos que según aumenta la complejidad musical disminuye la calidad del acompañamiento.

Por otra parte, según aumenta la complejidad del dominio, aumenta la calidad de los acompañamientos.

Para el problema de Bach todos los acompañamientos han sido buenos (de dominios con conocimiento). En el Apéndice I [10] se puede encontrar la planificación para el dominio Clásico de esta melodía.

La cucaracha se ve bien acompañada en los dominios clásico y completo mientras que un poco menos en el dominio de iniciación. La chispa adecuada no llega a tener un acompañamiento totalmente bueno ni siquiera con el dominio completo.

7 Conclusión

Como se ha demostrado en las pruebas, todos los objetivos iniciales del proyecto han sido cumplidos satisfactoriamente.

En primer lugar se ha conseguido representar una melodía musical utilizando un lenguaje de planificación. La melodía es el problema de entrada a nuestro planificador.

También se ha conseguido plasmar un conjunto de reglas armónicas en forma de acciones de un dominio.

Para ambos se ha utilizado el lenguaje PDDL. Y por medio de la ejecución del planificador LPG, se ha obtenido un plan que se traduce en el acompañamiento a la melodía.

No sólo esto, si no que además el proyecto implementa un sistema de generación de dominios dinámico a partir de una selección de acordes sobre una base de datos modificable y expansible.

Por otra parte, hemos conseguido generar los problemas a partir de ficheros de descripción musical, esto es MusicXML, al que se puede exportar desde MIDI automáticamente. Bien cierto es que esa traducción primera de MIDI a MusicXML no está implementada dentro del proyecto, mas también sobrepasa el ámbito de la obtención de planes que era la meta inicial. El esfuerzo en esta traducción habría sido injustificado en proporción al resto del desarrollo y se contaba con la ventaja de la existencia de herramientas que ya implementaban dicha traducción.

Más aún, a partir de todos los elementos de nuestra planificación (dominio, problema, plan y base de datos) se ha obtenido una salida en formato MIDI. Esto es un factor clave a la hora de evaluar la calidad de los planes que generaba el proyecto.

Todo lo anterior se automatiza dentro de un GUI que facilita y acelera desde la gestión de acordes de la base de datos, la selección para las acciones del dominio, la importación desde MusicXML, la ejecución del planificador y la exportación a MIDI.

A pesar de haber logrado los objetivos iniciales, el proyecto no está exento de limitaciones. La importación desde el formato MusicXML está restringido a melodías de unas características determinadas, también es necesario conocer la tónica de la pieza para poder realizar una importación correcta. El problema no puede tener notas alteradas, debe estar en compás cuatro por cuatro.

El formato de las acciones del dominio es limitado y sus precondiciones se centran únicamente en la nota a acompañar y no en el entorno. Esto es una de las razones por las cuales algunos acompañamientos (la chispa adecuada) no terminaban de cuadrar perfectamente. Tampoco se tienen en cuenta factores como la posición en el compás.

Inherentemente a las restricciones anteriores, el GUI está enfocado para no ir más allá de lo definido y por ello no dispone de herramientas para definir acciones en los acordes más complejas.

8 Líneas Futuras

Como solución a las limitaciones comentadas en el apartado de conclusiones y proyección futura de las directrices del proyecto, durante el desarrollo surgieron conceptos que por motivos técnicos o temporales no pudieron ser implementados.

Uno de ellos es el enriquecimiento de las reglas armónicas. Por medio de la modificación/expansión de las precondiciones de las acciones del dominio se podrían obtener reglas mucho más efectivas y que tuviesen en cuenta muchos más factores a parte de la nota individual. Esto se traduciría en planes mucho más agradables y adecuados. La familia de acordes utilizados para las pruebas podría ser ampliada infinitamente.

Paralelamente la interfaz gráfica debería ser capaz de gestionar estos nuevos tipos de precondiciones y de generar los dominios dinámicamente acorde a ellas.

La traducción de MusicXML a Problemas podría ser mejorada salvando las presentes limitaciones en cuanto a las expresiones en el formato MusicXML y a las propiedades de las piezas en particular.

La generación de la salida MIDI podría implementar elementos como la selección de instrumentos o la inclusión de varios planes distintos de acompañamiento para una misma pieza por instrumentos distintos.

9 Bibliografía

[Hil]: L. Hiller, L. Isaacson, Musical Composition with a High-Speed Digital Computer, 1993, from Machine Models of Music by S. M. Schwanauer, D. A. Levitt, MIT Press, 9-21

[Sch]: S. M. Schwanauer, D. A. Levitt, Machine Models of Music, 1993,MIT Press,

[Moo]: J. A. Moorer, Music and Computer Compostion, 1993, from Machine Models of Music by S. M. Schwanauer, D. A. Levitt, MIT Press, 167-186

[Lev]: D. A. Levitt, A Representation for Musical Dialects, 1993, from Machine Models of Music by S. M. Schwanauer, D. A. Levitt, MIT Press, 455-469

[Rad]: G. M. Rader, A Method for Composing Simple Traditional Music by Computer, 1993, from Machine Models of Music by S. M. Schwanauer, D. A. Levitt, MIT Press, 243-260

[Sim]: H. A. Simon, R. K. Sumner, Patterns in Music, 1993, from Machine Models of Music by S. M. Schwanauer, D. A. Levitt, MIT Press, 83-110

[Ler]: F. Lerdahl, R. Jackendoff, An Overview of Hierarchical Structure in Music, 1983, Music Perception,229-252

[Min]: M. Minsky, Music, Mind and Meaning, 1993, from Machine Models of Music by S. M. Schwanauer, D. A. Levitt, MIT Press, 327-354

[Rot]: J. Rothgeb, Simulating Musical Skills by Digital Computer, 1993, from Machine Models of Music by S. M. Schwanauer, D. A. Levitt, MIT Press, 157-164

[Ebc]: K. Ebcioglu, An Expert System for Harmonizing Four-Part Chorals, 1993, from Machine Models of Music by S. M. Schwanauer, D. A. Levitt, MIT Press, 385-401

[Bha]: J. Bharucha, MUSACT: A Connectionist Model of Musical Harmony, 1993, from Machine Models of Music by S. M. Schwanauer, D. A. Levitt, MIT Press, 497-509

[Feu]: J. Feulner, Neural Networks that Learn and Reproduce Various Styles of Harmonizations, 1993,Proceedings of the 1993 International Computer Music Conference , Computer Music Association

[Hor01]: D. Hörnel, P. Degenhardt, A Neural Organist Improvising Baroque Style Melodic Variations, 1997,Proceedings of the 1997 International Computer Music Conference , International Computer Music Association

[Hor02]: D. Hörnel, W. Menzel, Learning Musical Structures and Style with Neural Networks, 1998, Journal on New Music Research,44-62

[Pac]: F. Pachet, P. Roy, Formulating Constraint-Satisfaction Problems on PartWhole Relations: The Case of Automatic Harmonization, 1998, Paper presented at the ECAI'98 Workshop on Constraint Techniques for Artistic Application

[Sab]: J. Sabater, J. L. Arcos, R. López de Mántaras, Using Rules to Support Case-Based Reasoning for Harmonizing Melodies, 1998, Paper presented at the AAAI Spring Symposium on Multimodal Reasoning

[Mor]: R. Morales-Manzanares, E. F. Morales, R. Danenberg, J. Berger, SICIB: An Interactive Music Composition System using Body Movements, 2001, Journal of New Music Research,25-36

[Cop01]: D. Cope, Pattern Matching as an Engine for the Computer Simulation of Musical Style, 1990,Proceedings of the 1990 International Computer Music Conference , International Computer Association

[Cop02]: D. Cope, Experiments in Music Intelligence, 1987,Proceedings of the 1987 Computer Music Conference , International Computer Music Associations

[Fry]: C. Fry, FLAVORSBAND: a Language for Specifying Musical Style, 1993, from

Machine Models Of Music by S. M. Schwanayer, D. A. Levitt, MIT Press, 427-451
[Bil]: J. A. Biles, GENJAM: A Genetic Algorithm for Generating Jazz Solos, 1994
[Pap]: G. Papadopoulos, G. Wiggins, A Genetic Algorithm for The Generation of Jazz Melodies, 1998, Paper presented at the Finish Conference on Artificial Intelligence (SteP'98)
[Fra]: J. A. Franklin, Multiphase Learning for Jazz Improvisation and Interaction, 2001, Paper presented at the Eighth Biennial Symposium on Art and Technology, 1–3 March, New London, Connecticut.
[Tho]: B. Thom, BOB: An Improvisational Music Companion, 2001, Carnegie-Mellon
[Joh02]: P. N. Johnson-Laird, Jazz Improvisation: A Theory at the Computational Level, 1991,,
[Dan01]: R. B. Dannenberg, Software Design for Interactive Multimedia Performance, 1993, Interface,213-218
[Wes]: D. Wessel, M. Wright, S. A. Kahn, Preparation for Improvised Performance in Collaboration with Khyal singer, 1998
[Baa]: B. Baars, A Cognitive Theory of Consciousness, 1998,Cambridge University Press,
[Joh01]: M. L. Johnson, An Expert System for the Articulation of Bach Fugue Melodies, 1992,IEEE Computer Society,41-51
[Bre01]: R. Bresin, Articulation Rules for Automatic Music Performance, 2002,Proceedings of the 2001 International Computer Music Conference , International Computer Music Association
[Fri01]: A. Friberg, A Cuantitative Rule System for Music and Performance, 1995,
[Fri02]: A. Firberg, R. Bresing, L. Fryden, J. Sunberg, Musical Punctuation on the Microlevel: Automatic Identification and Performance of Small Melodic Units, 1998, Journal of New Music Research,271-292
[Fri03]: A. Friberg, J. Sunberg, L. Fryden, Music from Motion: Sound Level Envelopes of Tones Epressing Human Locomotion, 2000, Journal of New Music Research,199-210
[Wid01]: G. Widmer, The Musical Expression Project: A Challenge for Machine Learning and Knowledge Discovery, 2001, ,
[Wid02]: G. Widmer, Learning Expressive Performance: The Structure-Level approach, 1996, Journal of New Music Research,179-205
[Can]: S. Canazza, G. De Poli, A. Roda, A. Vidolin, Analysis and Synthesis of Expressive Intention in a Clarinet Performance, 1997,Proceedings of the 1997 International Computer Music Conference ,
[Dan02]: R. B. Dannenberg, I. Derenyi, Combining Instruments and Performance Models for High Quality Music Synthesis, 1998, Journal of New Music Research,211-238
[Bre02]: R. Bresin, Artificial Neural Networks–Based Models for Automatic Performance of Musical Scores, 1998, Journal of New Music Research,239-270
[Ken]: R. A. Kendall, E. C. Carterette, The Communication of Musical Expression, 1990, Music Perception,129
[Lop]: R. López de Mántaras, J. L. Arcos, From Composition to Expressive Performance, 2006, AI Magazine,43-57
[Mid01]: , Musical Instrument Digital Interface, 2008, http://en.wikipedia.org/wiki/Musical_Instrument_Digital_Interface
[Son]: , , 2007, http://www.sonicspot.com/guide/midifiles.html
[McK]: Cory McKay, Automatic Genre Classification of MIDI Recordings, 2004,
[Cun]: S. Cunningham, Suitability of MusicXML as a Format for Computer Music Notation

& Interchange, , www.newi.ac.uk/cunninghams/research/SuitabilityMusicXML.pdf
[Exp]: , , , http://explanation-guide.info/meaning/MusicXML.html
[Her]: Enric Herrera, Teoría Musical y Armonía Moderna Vol. I, 1995,Antoni Bosch editor S.A.,34-46
[Lpg]: , Short description of LPG 1.2, 2003, http://zeus.ing.unibs.it/lpg/
[NBe]: , NetBeans Home Page, , http://www.netbeans.org/

10 Apéndice I. Acompañamiento Clásico.

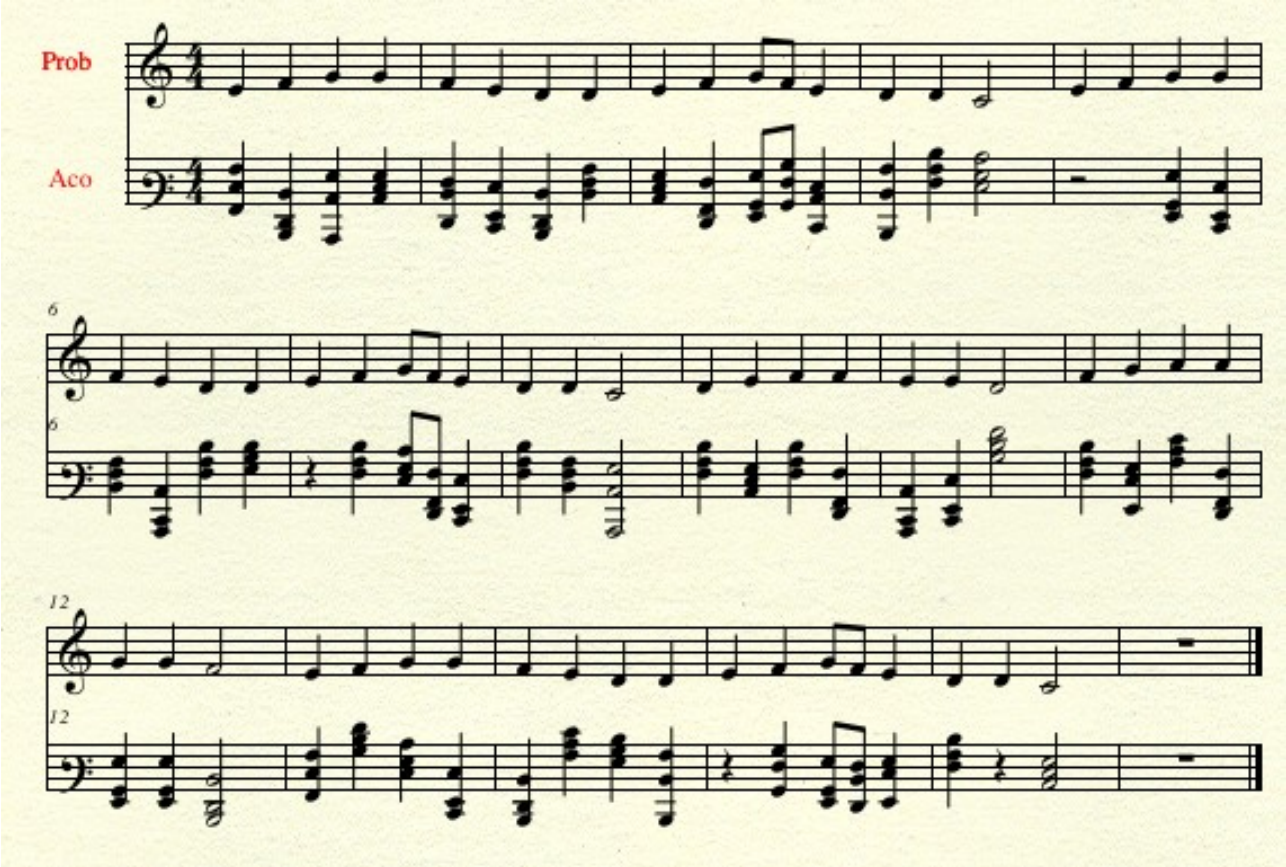
El siguiente plan corresponde a la salida del planificador para la obra de J. S. Bach, Werde munter mein gemüter, obtenida con el dominio Completo.

```
; Version LPG-td-1.0
; Seed 60926154
; Command line: lpg-td/lpg-td-1.0 -o /home/cr/Escritorio/Clasico/domain
-inst_with_contraddicting_objects -f /home/cr/Escritorio/pruebas/bach/bach
-speed -out /home/cr/Escritorio/pruebas/bach/Clasico1/clasico-plan
; Problem /home/cr/Escritorio/pruebas/bach/bach
; Actions having STRIPS duration
; Time 0.15
; Search time 0.09
; Parsing time 0.04
; Mutex time 0.01
; MetricValue 63.00

0:   (I_1 N1 N0) [1]
1:   (IV_1 N2 N1) [1]
2:   (I_2 N3 N2) [1]
3:   (I_2 N4 N3) [1]
4:   (IV_1 N5 N4) [1]
5:   (S_2 N6 N5) [1]
6:   (II_0 N7 N6) [1]
7:   (S_1 N8 N7) [1]
8:   (I_1 N9 N8) [1]
9:   (VII_1 N10 N9) [1]
10:   (S_4 N11 N10) [1]
11:   (S_3 N12 N11) [1]
12:   (VI_1 N13 N12) [1]
13:   (II_0 N14 N13) [1]
14:   (V_0 N15 N14) [1]
15:   (I_0 N16 N15) [1]
16:   (VI_1 N17 N16) [1]
17:   (S_3 N18 N17) [1]
18:   (V_1 N19 N18) [1]
19:   (S_4 N20 N19) [1]
20:   (IV_1 N21 N20) [1]
21:   (I_1 N22 N21) [1]
22:   (VII_0 N23 N22) [1]
23:   (II_0 N24 N23) [1]
24:   (III_0 N25 N24) [1]
25:   (IV_1 N26 N25) [1]
26:   (S_4 N27 N26) [1]
27:   (S_3 N28 N27) [1]
28:   (I_1 N29 N28) [1]
29:   (II_0 N30 N29) [1]
30:   (V_0 N31 N30) [1]
31:   (VI_0 N32 N31) [1]
32:   (II_0 N33 N32) [1]
33:   (I_1 N34 N33) [1]
34:   (II_1 N35 N34) [1]
35:   (II_1 N36 N35) [1]
36:   (VI_1 N37 N36) [1]
37:   (I_1 N38 N37) [1]
38:   (S_1 N39 N38) [1]
39:   (II_1 N40 N39) [1]
40:   (I_2 N41 N40) [1]
41:   (VI_2 N42 N41) [1]
42:   (VI_2 N43 N42) [1]
43:   (V_1 N44 N43) [1]
44:   (S_4 N45 N44) [1]
```

```
45:    (II_1 N46 N45) [1]
46:    (I_1 N47 N46) [1]
47:    (S_3 N48 N47) [1]
48:    (III_1 N49 N48) [1]
49:    (I_2 N50 N49) [1]
50:    (IV_1 N51 N50) [1]
51:    (S_2 N52 N51) [1]
52:    (S_1 N53 N52) [1]
53:    (II_0 N54 N53) [1]
54:    (S_2 N55 N54) [1]
55:    (S_3 N56 N55) [1]
56:    (I_2 N57 N56) [1]
57:    (S_3 N58 N57) [1]
58:    (III_0 N59 N58) [1]
59:    (II_0 N60 N59) [1]
60:    (V_0 N61 N60) [1]
61:    (I_0 N62 N61) [1]
62:    (FINALIZAR NLAST N62) [1]
```

La siguiente partitura corresponde a la melodía de la pieza junto con el acompañamiento del plan arriba especificado:



11 Apéndice 2. Manual de usuario.

Este documento explica cada uno de los elementos que componen el diseño de la interfaz gráfica de usuario de MusicMaker.
La interfaz implementa la siguiente funcionalidad:

- Gestión de una biblioteca de acordes.
- Importación de ficheros MusicXML.
- Creación de Dominios y Problemas PDDL según el formato de MusicMaker.
- Ejecución de un planificador para los anteriores dominio/problema.
- Generación de ficheros MIDI a partir del dominio/problema/planificación.

La interfaz se divide en tres partes: ChordEditor, SongMaker y Converter. Cada una de ellas se encarga de un proceso distinto; respectivamente, editar la base de datos de acordes, generar una planificación para las canciones y convertir las planificaciones en ficheros Midi.
Después de la explicación de cada uno de los elementos de la interfaz se encuentra una guía [4.] de cómo acompañar una melodía Midi guardada en formato MusicXML de principio a fin.

1. ChordEditor

Esta pestaña nos permite editar los acordes que se encuentran en la base de datos. También es en esta pestaña donde seleccionamos los acordes que el planificador utilizará en la siguiente pestaña para crear los acompañamientos.

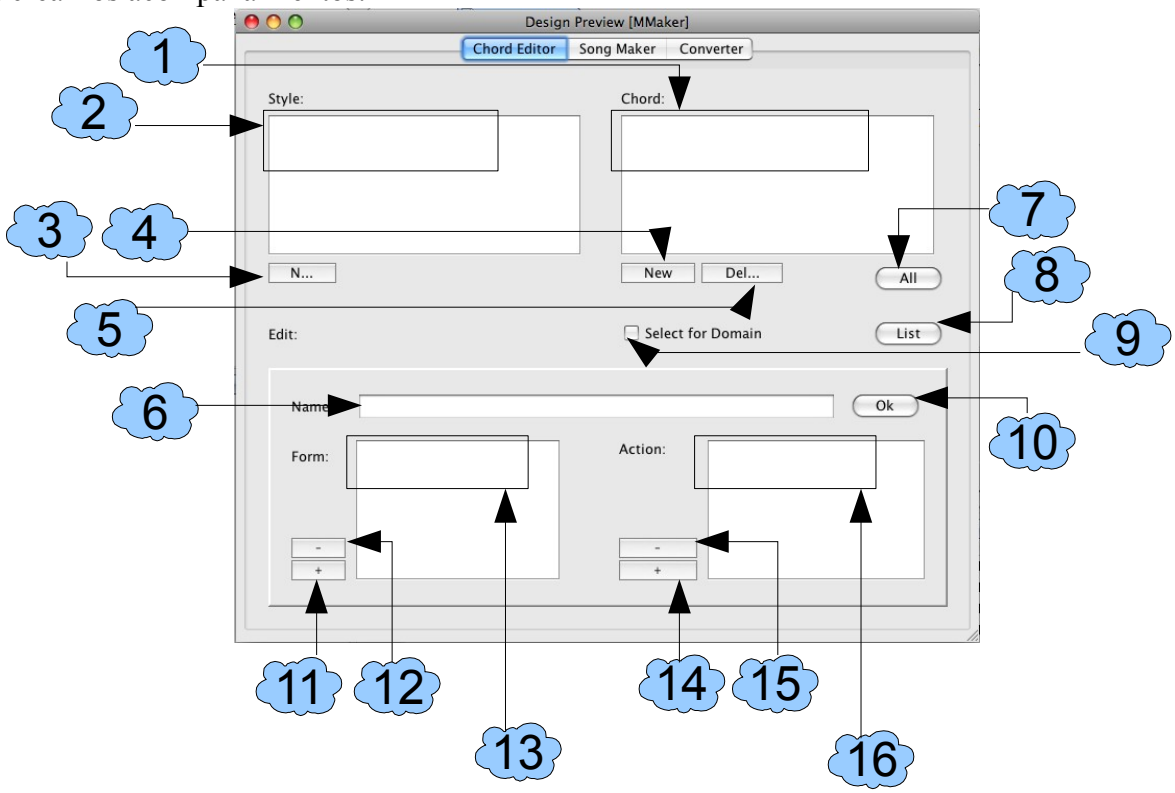


Figura 1. Ventana de MusicMaker en la pestaña ChordEditor.

1 Lista de acordes.

Aquí se muestran los acordes referidos al estilo seleccionado. Podemos seleccionar los acordes

de uno en uno para su inclusión en la lista que formará el dominio o para editar sus datos.

2 **Lista de estilos.**

Aquí se muestra la lista de estilos existentes en la base de datos. Se pueden elegir estilos de uno en uno, sus respectivos acordes aparecerán en la *Lista de acordes*. El estilo TODOS estará siempre presente y al seleccionarlo se mostrarán todos los acordes existentes en la base de datos.

3 **Crear nuevo estilo.**

Al seleccionar este botón insertamos un estilo nuevo en la base de datos. Un estilo nuevo no tendrá ningún acorde inicialmente.

4 **Crear nuevo acorde.**

Al seleccionar este botón creamos un acorde nuevo para el estilo seleccionado. Un acorde nuevo no tiene ninguna forma o acción especificada.

5 **Borrar acorde.**

Al seleccionar este botón eliminamos el acorde seleccionado de la base de datos. Al eliminar un acorde, automáticamente se eliminan sus formas y acciones.

6 **Nombre del acorde.**

Este campo editable representa el nombre del acorde seleccionado. Este campo es editable y para hacer persistente el cambio deberá seleccionarse el botón *Cambiar el nombre al acorde*.

7 **Seleccionar todos los acordes.**

Al seleccionar este botón todos los acordes de la base de datos serán utilizados en el dominio de planificación.

8 **Mostrar la lista de acordes seleccionados.**

Muestra un diálogo con una lista de los acordes que hemos seleccionado para formar parte del dominio de planificación.

9 **Seleccionar el acorde para el dominio.**

Muestra si el acorde seleccionado en la lista de acordes formará parte del dominio de planificación.

10 **Cambiar el nombre al acorde.**

Al seleccionar el botón cualquier cambio realizado en el campo *Nombre del acorde* será guardado.

11 **Nueva forma.**

Crea una nueva forma para el acorde seleccionado.

12 **Borrar forma.**

Elimina la forma seleccionada en la *Lista de formas*.

13 **Lista de formas.**

Lista de las formas que tiene el acorde. Una forma es cada una de las notas que forman el acorde de manera relativa a la raíz de éste, esto es p1.

14 **Nueva acción.**

Crea una nueva acción para el acorde seleccionado.

15 **Borrar acción.**

Elimina la acción seleccionada de la *Lista de acciones*.

16 **Lista de acciones.**

Lista de las acciones del acorde. Una acción es cada uno de los tonos a los cuales un acorde puede acompañar.

2. SongMaker

En esta pestaña es donde seleccionamos de dónde se carga el problema y dónde se guarda el dominio, los acordes y la salida del planificador para poder generar después una salida audible. Podemos importar en esta pestaña un fichero en formato MusicXML y guardarlo con el formato de problema, ejecutar el planificador y observar el resultado de tal ejecución.

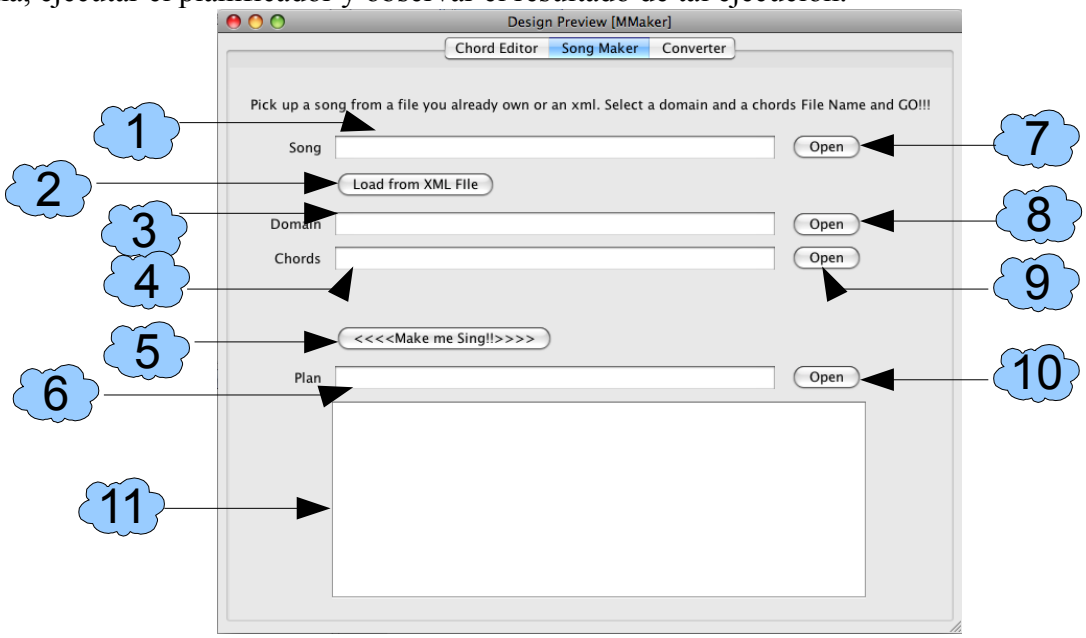


Figura 2. Ventana de MusicMaker en la pestaña SongMaker.

1 Nombre del fichero de la canción.

Ruta y nombre del fichero de problema que el planificador tomará como entrada.

2 Importar desde XML.

Importa desde un fichero en XML, los datos y los guarda en otro fichero de problema.

3 Nombre del fichero de dominio.

Ruta y nombre donde se guardará el dominio creado con la lista de acordes que se seleccionaron en la pestaña Chord Editor [1.]. El fichero luego será tomado como entrada por el planificador.

4 Nombre del fichero de acordes.

Ruta donde se guardará la definición de los acordes seleccionados en la pestaña Chord Editor [1.].

5 Ejecutar el planificador.

Al seleccionar el botón se ejecutará el planificador.

6 **Nombre del fichero de salida del planificador.**

Ruta y nombre del fichero donde se guardará el plan que ha obtenido el planificador en su ejecución.

7, 8, 9, 10 **: Abrir dialogo de selección de fichero**

Abre un dialogo de selección de archivo para cada uno de los campos a los que acompaña.

11 **Salida del planificador.**

Muestra la salida del planificador.

3. Converter

En esta pestaña seleccionamos la situación de los ficheros que hemos creado en la ejecución del planificador en la pestaña SongMaker [2.], para poder generar una salida Midi de distintos tipos. Obviamente también indicamos aquí la ruta de salida que deseamos para tales ficheros Midi.

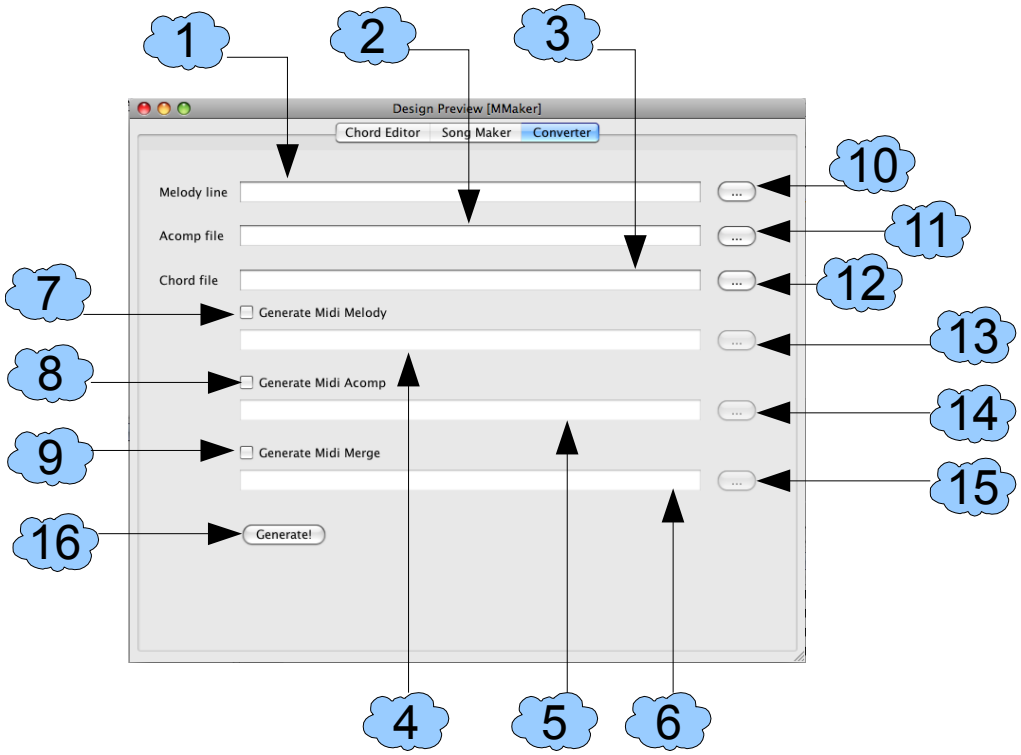


Figura 3. Ventana de MusicMaker en la pestaña Converter.

1 **Nombre del fichero con la melodía.**

Ruta y nombre del fichero que contiene el problema.

2 **Nombre del fichero con el acompañamiento.**

Ruta y nombre del fichero que contiene la salida del planificador.

3 **Nombre del fichero de acordes.**

Ruta y nombre del fichero de acordes.

4 **Nombre del fichero Midi de melodía.**

Ruta y nombre del fichero Midi donde se guardará la melodía generada.

5 **Nombre del fichero Midi de acompañamiento.**

Ruta y nombre del fichero Midi donde se guardará el acompañamiento generado.

6 **Nombre del fichero Midi completo.**

Ruta y nombre del fichero Midi donde se guardará tanto el acompañamiento como la melodía juntas.

7 **Casilla de selección de creación del fichero de melodía.**

Seleccionar si se desea generar el fichero de melodía.

8 **Casilla de selección de creación del fichero de acompañamiento.**

Seleccionar si se desea generar el fichero de acompañamiento.

9 **Casilla de selección de creación del fichero completo.**

Seleccionar si se desea obtener el fichero de mezcla.

10, 11, 12, 13, 14, 15: **Abrir diálogo de selección de fichero**

Al seleccionar estos botones se abre un diálogo de selección de fichero para el campo que acompañan.

16 **Crear los ficheros de salida**

Genera los ficheros Midi.

4. Guía Paso a Paso

La siguiente lista describe como acompañar una melodía guardada en formato MusicXML con MusicMaker.

1. Abrir la pestaña ChordEditor.
    1. Seleccionar todos los acordes (All).
2. Abrir la pestaña SongMaker
    1. Seleccionar importar XML (Load From XML File).
    2. Seleccionar el fichero XML del diálogo.
    3. Seleccionar el fichero donde se guardará el problema (melodía en formato PDDL).
    4. Seleccionar la ruta donde se guardará el dominio
    5. Seleccionar la ruta donde se guardarán los acordes.
    6. Seleccionar la ruta donde se guardará el plan.
    7. Ejecutar el planificador (Make Me Sing).
3. Abrir la pestaña Converter (los campos se autorellenarán)
    1. Comprobar las rutas de los ficheros de melodía, plan y acordes.
    2. Seleccionar generar merge
    3. Introducir la ruta y fichero de salida Midi
    4. Seleccionar Generate.

12 Apéndice 3. Manual de Instalación.

El siguiente documento contiene la serie de pasos que es necesario realizar para poder tener una versión funcional de MusicMaker.

1.- Introducción

La plataforma MusicMaker se compone de dos partes principales. Por un lado la Base de Datos que mantiene todos los datos de acordes necesarios para acompañar las canciones. El otro elemento principal es el interfaz gráfico (GUI) desde el cual se realizan todas las operaciones, tanto de gestión de los acordes como de generación e importación de melodías.

2.-Pasos

Para el correcto funcionamiento de toda la plataforma se necesita tener instalados y configurados los siguientes elementos:

- MySQL
- Java RunTime Environment
- MusicMaker

2.1.- Instalación MySQL y Base de Datos.

La instalación de la base de datos y la creación de usuarios y tablas y entradas en esta es el primer paso que hay que llevar a cabo a la hora de instalar MusicMaker. A continuación se detallan los pasos que hay que realizar para poder utilizar la aplicación.

2.1.1.- Instalación de MySQL

MySQL es una base de datos open source. Esto significa que está disponible en su sitio web (www.mysql.com) una versión de descarga gratuita. En primer lugar habrá que navegar en su sitio y descargar la versión correcta. La recomendada para MusicMaker es (mysql Ver 14.12 Distrib 5.0.45). Sin embargo cualquier versión posterior también funcionará.

Para más información sobre la instalación de MySQL consultar la documentación propia en el sitio web (www.mysql.com).

2.1.2.- Creación de usuarios y base de datos.

Para que MusicMaker tenga acceso a la base de datos por medio del conector JDBC es necesario que la base de datos tenga creado un usuario y una base de datos para la aplicación.

El primer paso es crear al usuario. Para esto se necesita entrar en mysql en modo monitor como administrador. Los datos del usuario que se tienen que crear son los siguientes:

Nombre: *musicmaker*

Contraseña: *musicmaker*

El comando de creación de un usuario en mysql es el siguiente:

CREATE USER 'musicmaker' IDENTIFIED BY 'musicmaker';

Ahora se debe crear una base de datos donde poner las tablas con los datos. El comando en mysql para hacerlo es el siguinte:

CREATE DATABASE music;

Ahora se debe otorgar al usuario musicmaker los permisos para interactuar con esa base de datos:

GRANT ALL ON 'music. TO musicmaker';*

2.1.3.- Ejecución de los scripts.

Una vez que se tienen los pasos anteriores completados; nuestro usuario y una base de datos donde trabajar con permisos asignados, se procede a crear el esquema. Para ello se necesita cargar en mysql los siguientes scripts:

table_creation.sql

load_chord_database.sql

Esto se puede hacer de múltiples maneras (consultar manual mysql). Por línea de comandos, se debería ejecutar la siguiente linea fuera de mysql:

cat table_creation.sql | mysql -umusicamker -pmusicmaker music;

cat load_chord_database.sql | mysql -umusicmaker -pmusicmaker music;

Por fin se tiene la base de datos configurada para poder ejecutar la aplicación. Además hay unos acordes básicos cargados dentro de ella.

2.2.- Instalación JRE.

La interfaz gráfica de la aplicación funciona con tecnología JAVA. En la mayoría de los casos, la máquina virtual está instalada ya en el ordenador. En caso de que ésta no se encuentre instalada, se debe acudir al sitio web de java y descargar la última versión.

Para más información acerca de la instalación visitar el sitio web (java.sun.com)

2.3.- Instalación de la aplicación

La aplicación viene empaquetada en un fichero jar. Se puede colocar el paquete en cualquier sitio del árbol de directorios de su computadora.

3.- Ejecución de la aplicación

Una vez ubicado el fichero jar, hay que lanzarlo. Hacer doble click sobre el fichero jar lo ejecutará. En caso de que la instalación Java haya pasado por alto algún punto y no sea completamente correcta, se puede ejecutar desde linea de comandos de la siguiente manera:

java -jar MusicMaker.jar