eman ta zabal zazu
Universidad del País Vasco
Euskal Herriko Unibertsitatea
BILBOKO INGENIARITZA ESKOLA
ESCUELA DE INGENIERÍA DE BILBAO

Universidad del País Vasco

Grado en Ingeniería Informática de Gestión y Sistemas de Información

Herramienta de conversión de notación de acordeón diatónico a partitura universal: Compilador y aplicación web

Trabajo Fin de Grado

Autor: Alvaro Luzuriaga Aguilar

Director: Iñigo Perona Balda

Co-director: Iñigo Mendialdua Beitia

Curso: 2020-2021 **Fecha**: 26 de julio de 2021

Resumen

El proyecto aquí documentado ha sido el resultado de la investigación y trabajo presentado como propuesta de solución, a un problema real existente en el mundo de la música. El problema en cuestión involucra a los 'trikitixalaris' y su específica manera de leer y escribir las partituras que después llegan a interpretar. Este tipo de notación llamado usualmente 'zenbakizko' no tiene ninguna relación con la escritura de partitura en pentagrama tradicional del que todos están acostumbrados.

Por tanto, mediante herramientas informáticas y del desarrollo del software, se ha desarrollado una herramienta capaz de suplir este problema de manera sencilla para todo usuario habituado a este lenguaje tan único de la 'trikitixa'.

Por consiguiente, la elaboración de dicha herramienta ha constado de la creación de un lenguaje intermedio comprensible y amigable para cualquier 'trikitilari'. Este lenguaje será la entrada del compilador, el cual se ha confeccionado a través de los analizadores Flex y Bison, con los cuales es posible procesar la léxica y gramática requerida para este proyecto.

El proceso de compilación da como resultado un archivo en notación Lilypond, pudiendo obtener a través del programa del mismo nombre una partitura de pentagrama detallada en formato PDF, PNG incluso audio MIDI.

Para que todo usuario tenga acceso a estas herramientas se ha valido del framework de desarrollo web Django, escrito en su mayoría en Python y HTML. Así se ofrecerá dicho compilador de manera clara a todo quien necesite de su uso.

La memoria se ha elaborado en el editor online Overleaf, editor colaborativo donde poder escribir documentos en LaTeX. Esta incluye una introducción al proyecto, la planificación y gestión del mismo, objetivos a cumplir y alternativas valoradas, el diseño y desarrollo del compilador y la web, así como las pruebas de los mismos.

Finalmente se incluyen apéndices detallando el significado de la notación 'zenbakizko' de la trikitixa, profundización en la notación Lilypond y la gramática del lenguaje intermedio.

Índice general

Índice de tablas

Índice de figuras

Glosario

Acorde Grupo de tres o más notas que juntas tocadas simultáneamente, que constituyen una unidad armónica. Estos son utilizados como acompañamiento para la melodía y para denotar la situación armónica de la obra.

Alteración Las alteraciones son signos empleados en la escritura musical, los cuales indican modificar la entonación o altura de cierta nota concreta. También llamados accidentes, los más usados son el «sostenido» ♯, «bemol» ♭ y «becuadro» ♮.

Árbol sintáctico Es la estructura que genera el analizador sintáctico de manera dinámica a medida que se efectúa el análisis sintáctico. Es una estructura estándar basada en un apuntador, pudiendo entonces conservar el árbol completo en una variable simple que apunte al nodo raíz.

Armadura La llamada armadura de tonalidad, o simplemente armadura es la agrupación de signos de alteración propias de una escala. Estas se sitúan al comienzo de del pentagrama, después de la clave. Su labor es situar la escritura musical empleada en ese pentagrama en cierta tonalidad.

Back-end Es uno de los dos componentes en los que se divide el proceso de compilación. Traduce al lenguaje de destino o ensamblador el lenguaje intermedio previamente traducido por el Front-end. Es también conocida como la parte de síntesis.

Bajo El bajo o línea de bajo, es la sucesión de sonidos musicales más graves de un pasaje. Su función es apoyar al resto de elementos de ese pasaje y fundamentar la progresión armónica.

Cifrado americano Tipo de escritura musical que se basa en el alfabeto para la representación de notas, acordes, escalas y tonalidades.

Clave En la notación musical se conoce como clave al símbolo ubicado en el inicio del pentagrama. La cual indica la altura correspondiente de las notas escritas en las líneas de ese mismo

pentagrama. Existen las claves de «sol» 𝄞, «fa» 𝄢, y «do» 𝄡.

Coda La coda en la música puede tener dos significados. El primero, como sección musical a modo de epílogo al final de un movimiento. Y el segundo, signo utilizado para indicar referencias de salto a la hora de realizar una repetición en una obra. El icono que indica el salto es 𝄋 y de donde a donde se tiene que saltar 𝄌.

Código intermedio En el ámbito de los compiladores, se considera código intermedio a cadenas de texto o archivos de texto de carácter temporal, que refleja el proceso de transformación de código fuente a código destino. Es necesario para subsanar posibles errores, que tenga una capacidad reorganizativa sencilla. No confundir con el 'código intermedio' creado para la escritura 'zenbakizko'.

Compás Entidad básica de la métrica musical, la cual está compuesta por grupos de unidades de tiempo. Estas unidades están divididas dentro del compás como acentuadas o átonas. En función del número de unidades de tiempo el compás puede ser binario, ternario o cuaternario. Siendo el número que aparece en la parte superior del compás el indicador del tipo de compás: **2**, **3**, **4** siendo respectivamente los tipos de compases principales.

Cromático Término musical utilizado para referirse tanto a instrumentos como escalas. Hace referencia a las notas intermedias de la escala o semitonos, que permanecen 'fijos' en la música diatónica. Se considera lo opuesto a diatónico. La escala cromática está compuesta por doce semitonos seguidos de orden ascendente.

Diatónico Termino musical utilizado para referirse tanto a instrumentos como escalas. Denotando la procedencia según una sucesión de sonidos naturales, es decir, los tonos y semitonos de la escala sin alteraciones cromáticas. Se considera lo opuesto a cromático. La escala diatónica está compuesta por doce sonidos de orden ascendente con el siguiente orden: *tono-tono-semitono-tono-tono-tono-semitono*.

Digitación Sistema de numérico (simbólico en la trikitixa) empleado en las escrituras musicales con con la intención de indicar al intérprete que dedo debe usar para tocar que especifica nota en el instrumento empleado.

Escala Conocida como escala musical, es el conjunto de sonidos ordenados con cierto sentido de conjunto sonoro. Dependiendo de la escala, este sentido varía, ya que depende de los intervalos existentes entre las notas de dicha escala. Las escalas más conocidas son las mayores (ej. Do Mayor: *do-re-mi-fa-sol-la-si*; *C-D-E-F-G-A-B*) y las menores (ej. La Menor: *la-si-do-re-mi-fa-sol*; *A-B-C-D-E-F-G*).

Front-end Es uno de los dos componentes en los que se divide el proceso de compilación. Se encarga en parserar el código fuente y convertirlo en el lenguaje intermedio comprensible por el compilador. Es también conocida como la parte de análisis.

Intensidad En la música se trata de la cualidad del sonido de la que se vale para diferenciar entre sonidos fuertes y suaves.

Intervalo Distancia o altura entre dos sonidos o notas musicales. Suele medirse en unidades de tonos y/o semitonos.

Melodía Sucesión de sonidos musicales diferentes en altura, la cual ordenada de manera coherente forman la unidad con sentido en la estructura musical llamada melodía. Esta puede ir acompañada o no por un acompañamiento o línea de bajo.

No-terminal En el ámbito de los compiladores, se refieren como conjuntos no-terminales a variables sintácticas que denotan conjuntos de cadenas de texto que ayudan a definir el lenguaje generado por la gramática. Con 'no-terminales' se refieren a que son un forma compleja del léxico utilizado en ese lenguaje, es decir que se puede simplificar en conjuntos más conjuntos terminales y/o no-terminales.

Parser Como parte de un compilador que es, el parser se encarga de determinar la estructura lógica de una porción de un texto, llamándose también analizador sintáctico.

Pentagrama Es la base donde se escriben las notas musicales y signos de puntuación. Esta formado por cinco líneas paralelas y equidistantes.

Registro Conjunto de sonidos que es capaz de reproducir un instrumento o persona al cantar. Se suele determinar con el sonido más agudo y más bajo que puede hacer sonar cada instrumento. Como ejemplo de registros conocidos se tiene de las voces del coro, en orden de más grave a más agudo: *Bajo, Barítono, Tenor, Contratenor, Contralto, Mezzosoprano, Soprano.*

Ritmo Fuerza o movimiento formado por la sucesión de ciertos sonidos. A lo largo de la melodía se presentan notas y silencios definidos por el ritmo de la misma. Es posible definir que el ritmo está compuesto por ciclos reiterados en intervalos de tiempo.

Semitono Se considera semitono a cada una de las dos partes, iguales o desiguales, en las que un intervalo de un tono puede ser dividido. Este es el menor de los intervalos reproducible entre notas consecutivas de una escala diatónica.

Signo de repetición Signos utilizados en la notación musical para indicar que cierta sección o apartado debe repetirse.

Silencio Es un signo de carácter musical, el cual denota pausa en una obra con una duración representa gráficamente por la propia figura.

Tabla de símbolos Estructura de datos que mantiene la información asociada a los identificadores. Estos identificadores son funciones, variables, constantes, tipos de datos etcétera. Esta tabla interactúa casi siempre en todas las fases del compilador: el rastreador o analizador léxico, el analizador sintáctico o el analizador semántico puede introducir identificadores dentro de la tabla. Las fases de optimización y generación de código utilizarán la información proporcionada por la tabla de símbolos.

Tempo Indica la velocidad a la que debe ejecutarse una obra musical. A diferencia del ritmo, el tempo es un valor numérico. Se puede determinar por pulsos por segundo, con el

metrónomo, indicadores de expresión rítmica como *moderato*, *adagio*, *presto* etc.

Terminal En el ámbito de los compiladores se refieren como no-terminales a símbolos básicos con los cuales las cadenas de texto son formadas. Con 'terminales' se refieren a que son la forma más básica del léxico utilizado en ese lenguaje y no se puede simplificar.

Token Salida producida por un analizador léxico después de procesar una entrada como el código fuente de un programa. Suele ser la primera fase de un compilador.

Tonalidad En la música se considera tonalidad a la implicación de una organización jerárquica para las relaciones entre distintas alturas de las notas, dependiendo de la consonancia sonora con respecto a la tónica de dicha tonalidad. Siendo esta tónica una nota, acorde y escala diatónica.

Tónica La tónica o nota tónica referencia al primer grado de la escala musical en un sistema tonal, definiendo entonces la tonalidad.

Tono Se puede considerar el tono como el intervalo equivalente a dos semitonos. También es el grado de elevación del sonido dependiente de las cantidades de vibraciones por segundo.

Tresillo Conjunto de tres notas de mismo valor de duración, las cuales se interpretan en el mismo tiempo correspondiente al valor de dos de ellas. Para indicar este tratamiento especial, se señala con un '3' encima o debajo del grupo.

CAPÍTULO 1

INTRODUCCIÓN

El tema del TFG (Trabajo Fin de Grado) aquí desarrollado ha sido presentado por los profesores Iñigo Perona e Iñigo Mendialdua. Su idea parte como iniciativa de acercar la escritura utilizada por un tipo de acordeonistas propio de Euskadi, los conocidos como «Trikitilaris», a la escritura musical más convencional, la de las partituras basadas en pentagramas. A su vez, los músicos más acostumbrados a las partituras de pentagrama, podrán acercarse a conocer este tipo de escritura propio de los trikitilaris.

Siendo el instrumento de estos intérpretes la conocida como «Trikitixa» (Figura 1.1), un tipo de acordeón pequeño que se lleva usando desde el siglo XIX en el País Vasco, y los «trikitilaris» los intérpretes instruidos en el uso manejo de este instrumento [1], estos son generalmente habituados a leer y ejecutar obras musicales con un tipo de caligrafía y simbología propia, que no tiene ninguna relación con otra escritura generalmente conocida.

Este tipo de lenguaje musical propio de la trikitixa, está basado principalmente en sus cualidades como instrumento de viento además de como acordeón diatónico. En la misma se tiene en cuenta el estado del fuelle (abierto o cerrado), el o los botones a pulsar, si se está leyendo melodía o bajo, la digitación de cada pulsación etc. Toda la gramática está basada en números para las notas, y símbolos para la digitación y demás signos. No existe ninguna relación con la ya conocida escritura clásica en pentagrama, más que ser ambas escrituras orientadas al ámbito musical. De hecho, esta escritura numérica, carece totalmente de expresiones de intensidad y definición rítmica. Por ello, los acordeonistas suelen deducir el ritmo a oído o escuchando previamente a otro intérprete tocarla.

Se da el caso entonces, de que instrumentistas de ambas disciplinas tengan que interpretar a la vez una obra. Si bien sonoramente pueden llegar a entenderse, existen las ocasiones en las que la falta de comunicación y comprensión escrita pueda darse. Asimismo muchos instrumentos folclóricos perduran porque han acertado en orquestase con otros instrumentos, y en este sentido poder escribirse en partitura basada en pentagramas y tener herramientas que faciliten esta escritura aporta a que perduren.

Figura 1.1: Trikitixa convencional

1.1. Motivación y objetivos

Con lo comentado previamente vemos que existe un problema real, a la hora de querer reunir trikitilaris con otro tipo de instrumentistas para querer interpretar cierto tipo de obras. En muchas ocasiones unos no son capaces de comprender el lenguaje escrito del otro y esta falta de comunicación puede llegar a entorpecer el propio ensayo de la obra. Esta incomprensión sucede dado a que los trikitilaris leen y escriben partituras de una manera particular, a la que se llamará escritura «numérica», ya que en euskera es conocida como «zenbakiz».

Al ver como este contratiempo puede llegar a entorpecer el aprendizaje y elaboración de una obra musical en el ámbito grupal entre músicos de distinta especialidad, se ha querido proponer una solución y una posible implementación para la solución de este asunto. Así, se ha querido elaborar una herramienta que pueda acercar ambas escrituras, partiendo de primera mano de la escritura «numérica» para obtener una conversión a la ya comentada partitura clásica en pentagrama (Figura 1.2).

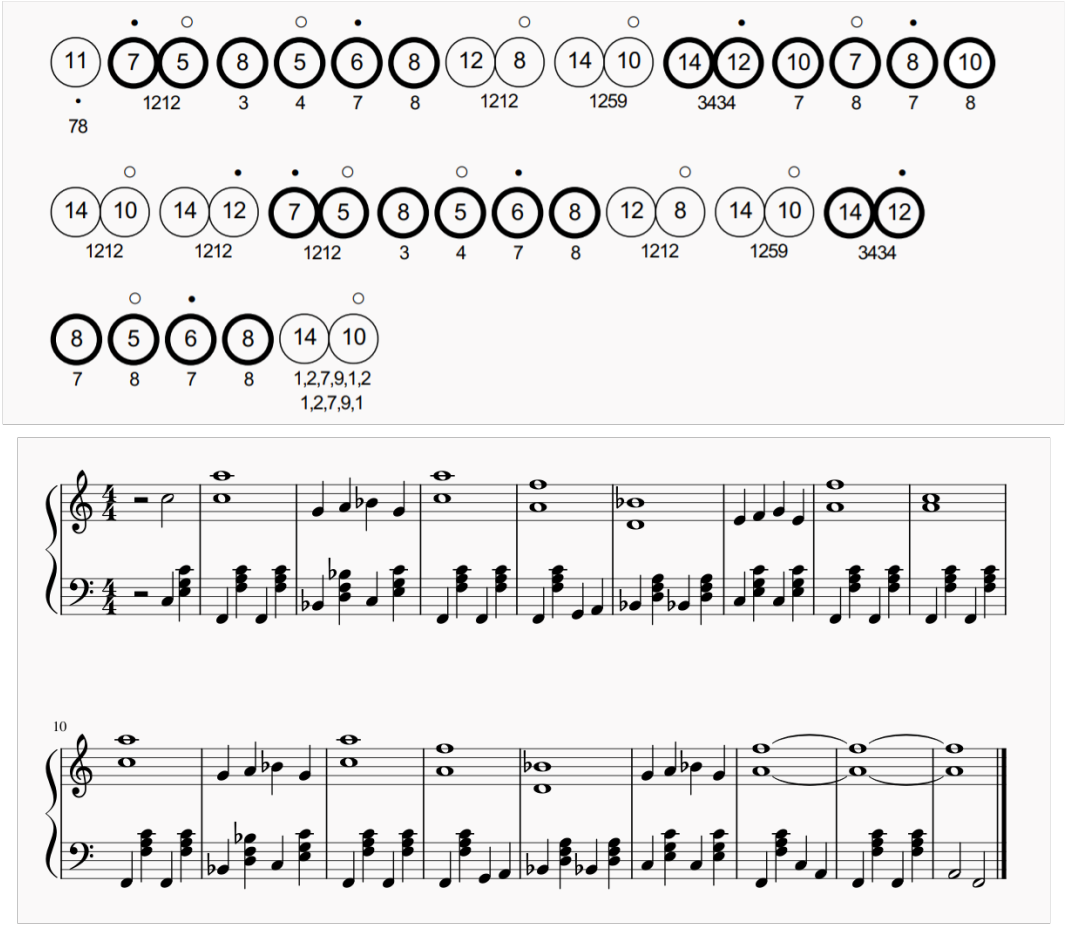


Figura 1.2: Comparación aproximada: escritura trikitixa - escritura pentagrama

Partitura: Ikusi mendizaleak

Además de que como pianista, me he encontrado en situaciones muy parecidas a la hora de acompañar a un instrumentista o tocar en conjunto con algún grupo. Puedo decir de primera mano que es un contratiempo real que solucionado, es muy posible que ahorre tiempo y esfuerzo a todos los que lleguen a encontrarse con esta situación.

Por ello se ha querido establecer dos objetivos principales para poder llegar a solventar la situación aquí expuesta. Siendo estos objetivos un compilador para la transformación de escritura numérica o «zenbakiz» a partitura tradicional. Y una página web accesible a todo el que esté interesado en usar esta herramienta de manera sencilla para cualquier usuario.

1. Compilador: tiene la función de procesar un lenguaje intermedio ideado para asemejarse a la escritura numérica, en un tipo de notación capaz de mostrar partituras en pentagramas.
2. Página web: es la plataforma de la que se valdrán los usuarios para interactuar con el compilador de diversas formas.

Los objetivos más detallados y específicos vienen definidos en el Capítulo 3.1

1.2. Planteamiento del problema

A fin de encontrar una resolución a este problema, se desarrollará un lenguaje intermedio. Este lenguaje quiere asemejarse lo máximo posible a lo que sería una partitura de trikitixa pero representada en un fichero de texto, ya que nuestro objetivo es que cualquier usuario dada una sencilla guía pueda usarlo sin mayores inconvenientes. Para ello, se partirá desde la página web de Trikitirauki[1], la cual es una web de dominio público en la que es posible crear, visualizar y compartir partituras de este tipo de escritura «numérica». Analizando dichas partituras se desarrollará dicho lenguaje con su propia estructura y gramática.

Se creará y definirá un compilador por medio de Flex y Bison. Para poder tratar este lenguaje y que dé como resultado una partitura convencional de pentagrama y así poder aunar ambas escrituras. Así se podrá convertir este «Lenguaje intermedio» desarrollado a una notación de lenguaje pensada para partituras llamada Lilypond (Figura 1.3). Al obtener el código en lenguaje Lilypond, el mismo puede realizar una transformación a PDF, obteniendo así la partitura en pentagrama. Esta intentará parecerse lo máximo posible a lo que se obtendría pasando a mano de una partitura numérica a una de pentagrama.

Finalmente se ideará una sencilla web con Django. En ella el propio usuario escribirá con el lenguaje intermedio la partitura que quiera obtener basado en números, mostrando la web el resultado de la compilación y pudiendo obtener la partitura en formato PDF. Esta partitura representará el resultado en pentagrama en notación numérica, junto a la notación universal basada en el pentagrama.

Este último apartado es importante, ya que uno de los mayores objetivos de este trabajo es acercar las soluciones aquí elaboradas a todo usuario no habituado al uso de aplicaciones informáticas. Así el usuario no deberá complicarse en instalar las dependencias del compilador, facilitando su uso en gran medida.

[1] https://tirikitrauki.com/zenbakiz/

Figura 1.3: Herramientas principales del planteamiento

1.3. Estructura de la memoria

Esta memoria contará con los siguientes capítulos:

- **Planteamiento inicial** (Capítulo 2): Apartado donde se expone el planteamiento inicial del proyecto. Se enunciarán las ideas organizativas originales del trabajo, para así conllevar a su correcta ejecución dentro de un tiempo estipulado con unos requisitos concretos.
- **Captura de requisitos** (Capítulo 3): Listado y justificación de los apartados a cumplir que tendrá el trabajo a realizar, así como las necesidades a satisfacer. Se justifica el uso de las herramientas escogidas y cuales se descartaron. También se exponen las inspiraciones tomadas para la elaboración de la web.
- **Análisis y diseño** (Capítulo 4): Explicación del diseño del proyecto, sus partes y cómo se divide. Principalmente orientado a la web, su estructura e interacción con el compilador.
- **Compilador** (Capítulo 5): En este apartado se explica las decisiones tomadas respecto al compilador, su diseño e implementación. Esto es: el léxico y gramática del lenguaje intermedio creado, la estructura del compilador y el proceso de su ejecución.
- **Web: Django** (Capítulo 6): Desarrollo de la web en Django, cuales son sus ficheros y funcionalidades principales, así como una explicación de su implementación.
- **Pruebas** (Capítulo 7): Definición de las pruebas realizadas con sus entradas y salidas, así como la justificación del resultado obtenido y la relevancia del mismo.
- **Conclusiones** (Capítulo 8): Es la recapitulación del trabajo elaborado y de los objetivos cumplidos. Se concluirá comentando el transcurso del proyecto. Aclaración de cómo se debe realizar el trabajo una vez conocido sus puntos críticos. Qué posibles mejoras

o añadidos se podría realizar en un futuro, así como la valoración de estas. Se realizará un análisis crítico del resultado final y de su utilidad. Así como un comentario sobre la experiencia personal.

Además esta memoria cuenta con los siguientes apéndices:

- **Escritura de trikitixa** (Apéndice A): En el primer apéndice se detalla la estructura de la escritura propia de la trikitixa y que significa en una notación musical clásica en pentagrama.
- **Lilypond** (Apéndice B): Se detalla el significado de la notación utilizada en los ficheros de Lilypond, así como su significado en una notación musical clásica en pentagrama.
- **Gramática del lenguaje intermedio** (Apéndice C): Demostración de la gramática utilizada en el fichero de Bison, su estructura así como los nodos terminales y no terminales.

CAPÍTULO 2

PLANTEAMIENTO INICIAL

Mediante la siguiente sección se explicará como ha sido el transcurso del proyecto. Se expondrán la estructura y herramientas utilizadas para el mismo, una planificación inicial teniendo en cuenta sus riesgos, así como la evaluación económica para determinar sus costes.

2.1. Estructura

La estructura del proyecto la forman la combinación del patrón MVT (Model View Template) correspondiente a Django el cual es una variación del patrón MVC (Model View Controler) y el compilador generado con Flex y Bison. El cliente tendrá acceso a la plantilla creada por Django, pudiendo acceder así a su lógica y poder obtener, por medio del compilador, la partitura correspondiente al texto escrito como entrada (Figura 2.1).

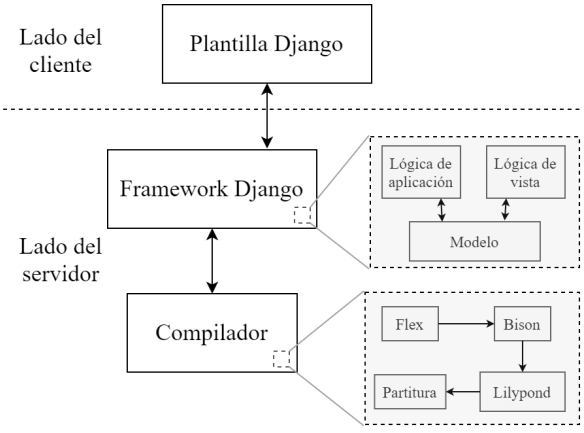


Figura 2.1: Estructura del proyecto

2.2. Herramientas y lenguajes

Se ha hecho uso de las siguientes herramientas y lenguajes para la elaboración del TFG. A continuación se detalla cuales han sido y su uso:

- **Atom**: Editor de código soportado en diversos sistemas operativos. En él se tiene la capacidad de instalar diversos plugins para acomodar la herramienta a la labor correspondiente. En este caso se han utilizado unos plugins de Flex y Bison para la edición de código.
- **Bison**: Bison es un generador de analizadores sintácticos que convirtiendo una descripción para una gramática independiente del contexto en un programa que analiza esta gramática en C. Su compatibilidad con Yacc es del 100 %, siendo Yacc una herramienta clásica de Unix para la generación de analizadores sintácticos.
- **Blackboard Collaborate**: Aplicación web provista por la UPV/EHU para poder impartir clases y tutorías entre profesores y alumnos. En este caso, para las reuniones periódicas de seguimiento.
- **Chrome**: Navegador de Google de código cerrado. Navegador principal para todos los servicios de Google.
- **CSS**: Es un lenguaje de diseño de hojas web. En este caso se ha usado junto HTML para el diseño de la página web.
- **Django**: Es un framework de código abierto para el desarrollo de páginas web orientadas al contenido. Escrito en Python, se basa en el diseño ya conocido como modelo–vista–controlador.
- **Draw.io**: Extensión de Google Drive con el que se pueden realizar de manera sencilla diagramas. En este caso para la creación de estructuras y arquitecturas de datos.
- **Flex**: Flex es una herramienta generadora de analizadores léxicos rápidos. Su compatibilidad con Lex es del 100 %, siendo Lex el analizador léxico estándar en los sistemas Unix.
- **GanttProyect**: Software de código abierto, orientado principalmente a la planificación de tareas temporal del proyecto.
- **GCC**: GNU Compiler Collection es un compilador del lenguaje de programación C, utilizado para la definición de métodos en Flex y Bison, y GCC para su compilación.
- **Gimp**: Programa de edición de imágenes. Software libre y gratuito, formando parte del proyecto GNU.
- **Git**: Es el sistema de control de versiones más estandarizado en el ámbito de la informática y desarrollo del software. Se ha utilizado para llevar a cabo una gestión de las versiones de la web desarrollada en Django junto a la plataforma GitHub, donde poder acceder al repositorio via navegador.

- **Google Docs**: Servicio de Google, para la creación, compartición y edición de documentos en la nube.
- **Google Drive**: Servicio de Google, orientado al alojamiento de ficheros y archivos compartidos en la nube.
- **HTML**: Es un lenguaje de marcas orientado para la elaboración de páginas web. Los navegadores lo interpretan y muestran en pantalla el resultado.
- **LaTeX**: Lenguaje orientado a la creación de textos de tipo marcado. Utilizado como fuente de la documentación.
- **Lilypond**: Programa de código abierto. Utiliza su propio lenguaje con notaciones sencillas para poder crear y editar partituras con gran detalle. Empleado para la creación de partituras en PDF.
- **MIDI**: La «Interfaz Digital de Instrumentos Musicales» es el lenguaje que permite a hardware e instrumentos musicales comunicarse entre ellos. También es un protocolo, en el que se incluye una interfaz y un lenguaje para transmitir los datos MIDI.
- **Outlook**: Servicio de mensajería electrónica de Microsoft. Es el utilizado ahora mismo por la UPV/EHU para la comunicación entre docente y alumno.
- **Overleaf**: Se ofrece como software de servicio en su propia página web [1]. Es un editor de documentos de Latex y está diseñado para proyectos colaborativos.
- **PDF**: Formato de almacenamiento de documentos mundialmente conocido y utilizado.
- **Python**: Lenguaje de programación, con soporte parcial a orientación de objetos. Destaca por su legibilidad de código y por los muchos ámbitos en el que se puede utilizar. En nuestro caso, nuestra web de Django está basado principalmente en este lenguaje.
- **Ubuntu**: Sistema operativo de distribución GNU/Linux, de software libre y código abierto. Virtualizado gracias a Virtualbox para hacer pruebas con el compilador.
- **Virtualbox**: Siendo un software de virtualización de arquitecturas de ordenadores tipo x86 y AMD64/Intel64, en las cuales permite instalar sistemas operativos. Se ha utilizado para desarrollar y hacer pruebas en un entorno GNU/Linux, para el apartado del compilador.
- **Visual Studio Code**: Es un editor de código abierto desarrollado por Microsoft, compatible con Windows, macOS y GNU/Linux. Se ha utilizado para el desarrollo de la web junto al framework Django, editando ficheros de Python y HTML.
- **Zenbakiz**: Esta web[2] creada para la creación y distribución de las partituras características de la trikitixa. Incluye un extenso catálogo de partituras numéricas y un editor propio.

[1] https://es.overleaf.com/
[2] https://tirikitrauki.com/zenbakiz/

2.3. Planificación

En el siguiente apartado se expone la planificación inicial propuesta para el proyecto.

Ciclo de vida

Para la ejecución de este proyecto se ha utilizado el desarrollo en cascada, también llamado desarrollo iterativo e incremental. Funciona de la siguiente manera: Las etapas se colocan una encima de la otra y se ejecutan de arriba hacia abajo, se ordenan de forma que para poder empezar una nueva fase tiene que estar terminada la anterior, por ejemplo, para poder hacer las pruebas tiene que estar hecha la implementación (Figura 2.2).

Estas son las fases a seguir:

- **Planificación y Gestión:** En esta primera fase se llevarán a cabo varias reuniones con el cliente, con el objetivo de planificar la manera en la que se va a desarrollar el proyecto. Desarrollando a su vez el DOP (Documento de Objetivos del Proyecto), la documentación correspondiente a los requisitos, objetivos y planificación de tareas para el proyecto. Indicando qué hay que desarrollar, cómo se va a desarrollar y quién lo va a desarrollar. Será necesario estimar la cantidad de tiempo a invertir, los posibles riesgos que puedan surgir y la distribución de trabajo.
- **Diseño:** En esta fase se definirá la forma en la que será implementada la aplicación teniendo en cuenta los requisitos del cliente y las restricciones del proyecto.
- **Implementación:** Se implementarán las funciones anteriormente diseñadas. Es posible que tenga que volverse a la fase anterior para redefinir posibles errores no previstos.
- **Pruebas y documentación:** Tras finalizar la implementación, se realizarán una serie de pruebas para asegurar el correcto funcionamiento de las funciones. Es posible que en esta fase también tenga que volverse a la anterior, para cubrir ciertos casos de prueba que no se han tenido en cuenta. Junto a ellas se irá documentando la parte práctica y teórica del proyecto, ya que según las pruebas se vayan completando será posible realizar una documentación sin errores.
- **Presentación:** Previamente al acto de presentar se preparará la memoria del proyecto y la presentación misma para así exponer correctamente el producto al cliente. Para concluir el proyecto se realizará la presentación al cliente con toda la documentación necesaria.

Como puede verse en la Figura 2.2, este tipo de ciclo de vida de proyecto da pie a una correcta organización del mismo, sin perder rumbo ni linealidad. Es posible volver a fases anteriores para realimentarlas, pero generalmente su estructura lineal es la que funciona bien con proyectos de este ámbito, más siendo un proyecto estable al estar desarrollado por un único individuo [2].

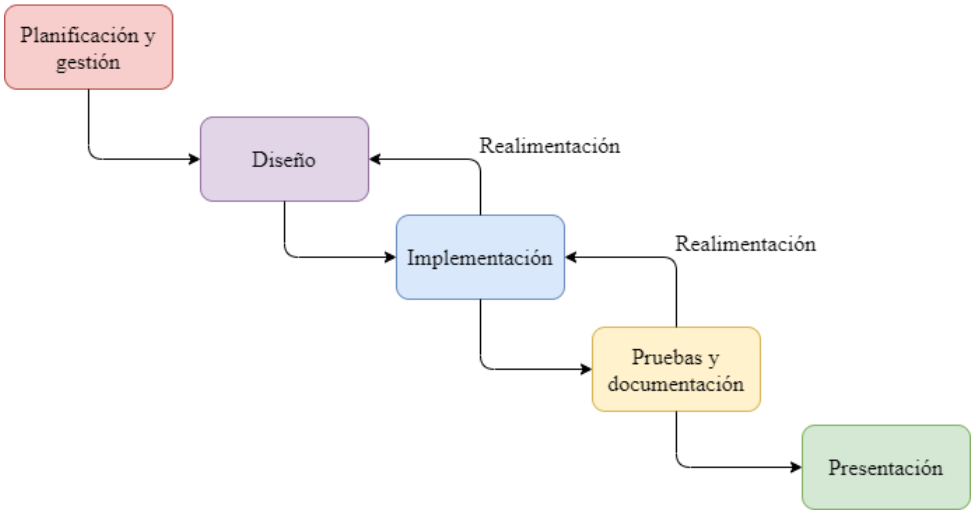


Figura 2.2: Ciclo de vida del proyecto

Diagrama EDT

Para definir el alcance del proyecto y descomponer jerárquicamente las tareas principales de las fases definidas en el ciclo de vida se utilizan los diagramas EDT (Eastern Daylight Time), sistemas de organización para la gestión de proyectos (Figura 2.3).

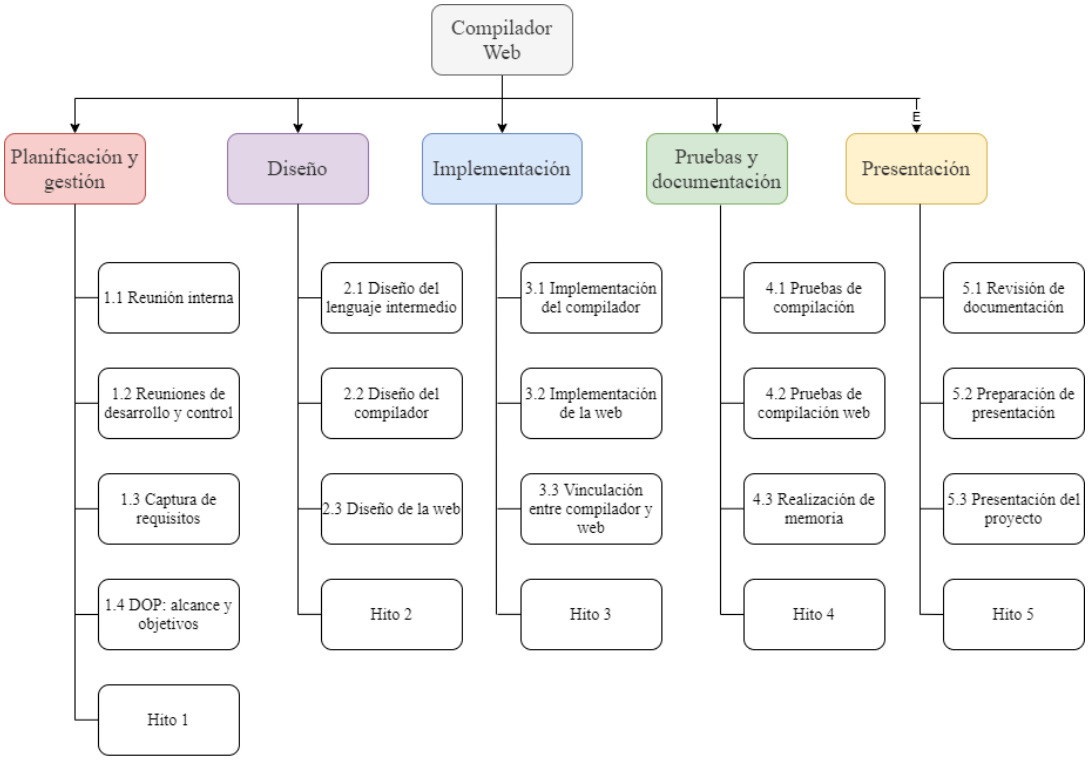


Figura 2.3: Diagrama EDT del proyecto

Planificación temporal

Para que el proyecto se elabore correctamente, dentro de un tiempo prefijado, se ha elaborado una tabla de tiempo, la cual recoge el esfuerzo de cada tarea, y de cada grupo de tareas (o fases). Así se deja claro cuál es la planificación temporal del proyecto. Cabe aclarar que se refiere a esfuerzo a la cantidad de horas reales invertidas en una tarea.

Fase/Tarea	Descripción	Esfuerzo
1. Planificación y gestión		62h
1.1 Reunión interna	Primeras reuniones con los interesados en el proyecto. Se expone el tema, objetivos principales y posibles soluciones a tomar. Se ponen en común opiniones sobre el mismo.	8h
1.2 Reuniones de desarrollo y control	Reuniones de seguimiento semanales a lo largo del proyecto para comprobar su estado. En estas se ponen en común el trabajo a realizar y el realizado, posibles correcciones o notificación de desvíos tomados.	26h
1.3 Captura de requisitos	La fase consta de determinar con los interesados del proyecto los requisitos a cumplir que tendrá el compilador y la web. Cuales serán necesarios y cuales sobran.	10h
1.4 Desarrollo del DOP	Elaboración del DOP, documento donde se indican el alcance, objetivos y planificación de tareas. Es posible encontrar este documento en el Capítulo 2.	28h

Tabla 2.1: Tabla de planificación temporal 'Planificación y gestión'

Fase/Tarea	Descripción	Esfuerzo
2. Diseño		49h
2.1 Diseño del lenguaje intermedio	Definir el léxico, gramática y sintaxis del que constará el lenguaje intermedio a introducir en el compilador, este tendrá que cubrir la mayor cantidad de aspectos del lenguaje 'zenbakiz' de la trikitixa posible.	16h
2.2 Diseño del compilador	Definir el compilador, su estructura funcionamiento y concreción de qué versión o alternativa se va a implementar.	13h
2.3 Diseño de la web	Definir la estructura y funcionamiento de la web. Así como sus funciones principales, prototipo de interfaz y su vinculación con el compilador.	20h

Tabla 2.2: Tabla de planificación temporal 'Diseño'

Fase/Tarea	Descripción	Esfuerzo
3. Implementación		89h
3.1 Implementación del compilador	Desarrollo del compilador completo, su manejo de errores y la relación con el lenguaje intermedio de origen y Lilypond como lenguaje de destino.	32h
3.2 Implementación de la web	Desarrollo de la aplicación web, su funcionamiento e interfaz.	38h
3.3 Vinculación entre compilador y web	Desarrollar el enlace entre la web y el compilador, para que con la interfaz de la primera se pueda acceder a la función de esta última.	19h

Tabla 2.3: Tabla de planificación temporal 'Implementación'

Fase/Tarea	Descripción	Esfuerzo
4. Pruebas y documentación		135h
4.1 Pruebas de compilación	Con el compilador ya desarrollado se han realizado pruebas convirtiendo partituras reales al lenguaje intermedio. Además se han creado pruebas propias para testear al sistema.	15h
4.2 Pruebas de compilación en la web	Una vez instalada la web, se ha probado su funcionamiento. Después se ha hecho lo propio con la web vinculada al compilador.	28h
4.3 Realización de la memoria	Elaboración de la memoria del proyecto; reflejando todos sus resultados, procesos, aciertos, fallos, y conclusiones tomadas.	92h

Tabla 2.4: Tabla de planificación temporal 'Pruebas y documentación'

Fase/Tarea	Descripción	Esfuerzo
5. Presentación		23h
5.1 Revisión de la documentación	Revisión de la documentación una vez examinada por terceros. Para así darle más empaque y uniformidad.	11h
5.2 Preparación de la presentación	Preparación de la presentación, tanto haciendo pruebas como realizando una demostración a los interesados de este proyecto.	10h
5.3 Presentación del proyecto	Presentación y defensa ante el tribunal sobre el proyecto aquí realizado.	2h
Total del proyecto		358h

Tabla 2.5: Tabla de planificación temporal 'Presentación'

Diagrama de Gantt

Se ha desarrollado un diagrama en 'GanttProject', el cual muestra la duración y precedencia de cada tarea. De esta manera, agrupando las tareas por fases y estimando una duración, es posible calcular aproximadamente la duración total del proyecto. Así como el correcto orden de ejecución de las tareas.

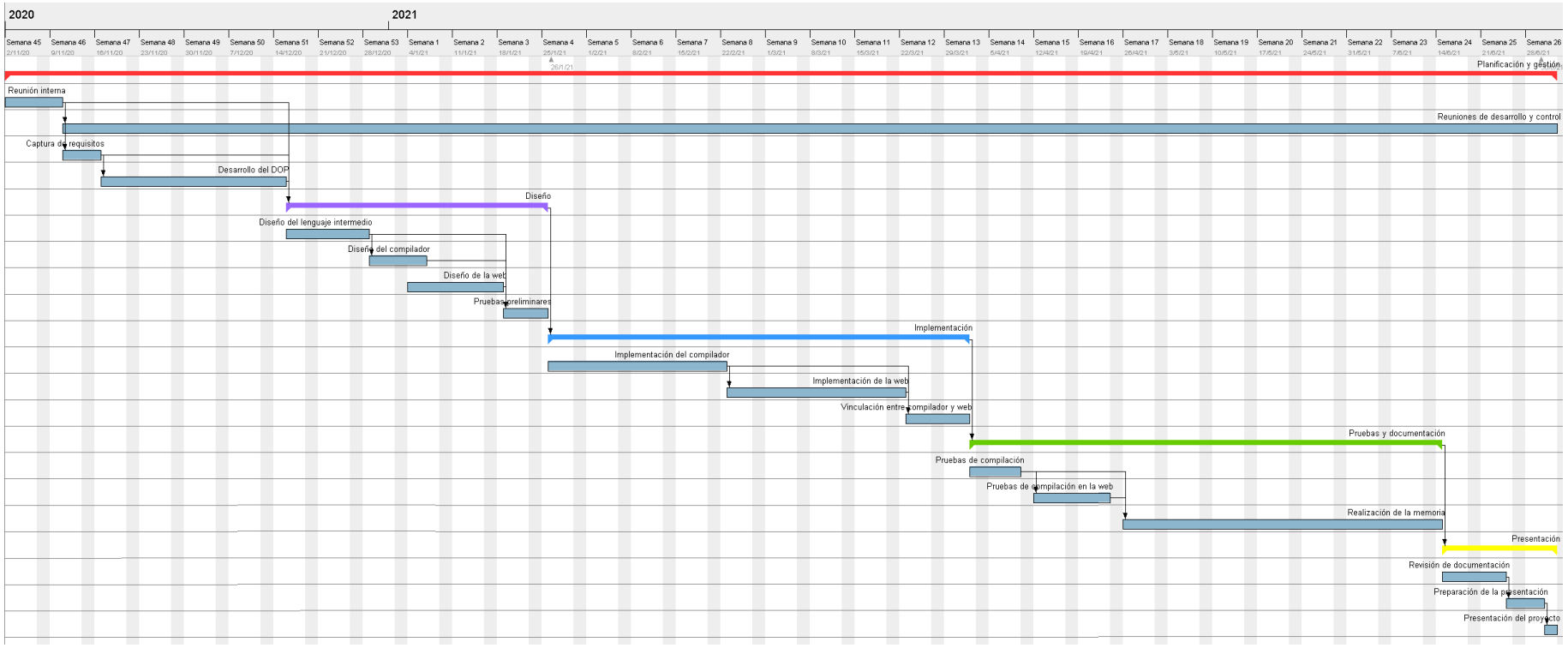


Figura 2.4: Diagrama Gantt de las tareas

Nombre	Fecha de inicio	Fecha de fin	Duración
Planificación y gestión	2/11/20	2/7/21	175
Reunión interna	2/11/20	10/11/20	7
Captura de requisitos	11/11/20	16/11/20	4
Reuniones de desarrollo y control	11/11/20	2/7/21	168
Desarrollo del DOP	17/11/20	15/12/20	21
Diseño	16/12/20	25/1/21	29
Diseño del lenguaje intermedio	16/12/20	28/12/20	9
Diseño del compilador	29/12/20	6/1/21	7
Diseño de la web	4/1/21	18/1/21	11
Pruebas preliminares	19/1/21	25/1/21	5
Implementación	26/1/21	1/4/21	48
Implementación del compilador	26/1/21	22/2/21	20
Implementación de la web	23/2/21	22/3/21	20
Vinculación entre compilador y web	23/3/21	1/4/21	8
Pruebas y documentación	2/4/21	14/6/21	52
Pruebas de compilación	2/4/21	9/4/21	6
Pruebas de compilación en la web	12/4/21	23/4/21	10
Realización de la memoria	26/4/21	14/6/21	36
Presentación	15/6/21	2/7/21	14
Revisión de documentación	15/6/21	24/6/21	8
Preparación de la presentación	25/6/21	30/6/21	4
Presentación del proyecto	1/7/21	2/7/21	2

Figura 2.5: Tabla de tiempos y duración del diagrama Gantt

El tiempo total estimado de desarrollo del proyecto son 358 horas. Teniendo esto en cuenta, invirtiendo 3 horas al día como media, 4 días a la semana durante 30 semanas obtendríamos las horas estimadas. Además teniendo en cuenta todo el margen de tiempo, podrán invertir más horas si son necesarias sin riesgos de no llegar a la fecha de entrega.

2.4. Gestión Riesgos

Existen ciertos riesgos a tener en cuenta. Sin una correcta gestión de riesgos las consecuencias en el transcurso del trabajo pueden ser graves. Por tanto se han identificado los siguientes riesgos de tres tipos de categorías.

Riesgos Técnicos

Requisitos: Cambio de objetivo	
Descripción	El objetivo del proyecto cambia, por requisito del cliente
Prevención	Se dejarán objetivos y alcance del proyecto claros desde un primer momento. Fijando los cambios factibles o añadidos en un posible futuro.
Plan de contingencia	Mantener la comunicación con el cliente.
Probabilidad	Baja.
Impacto	Alto.

Tabla 2.6: 'Actualización de requisitos' de los Riesgos Técnicos

Tecnología: Pérdida de progreso	
Descripción	Pérdida parcial o total del progreso del proyecto. Debido a un error de software o de hardware.
Prevención	Realización de copias de seguridad incrementales en un periodo acordado. Estas copias de seguridad se realizarán en distintos dispositivos y formatos.
Plan de contingencia	Tener una copia de seguridad anterior para la recuperación de trabajo.
Probabilidad	Baja.
Impacto	Alto.

Tabla 2.7: 'Pérdida de progreso' de los Riesgos Técnicos

Diseño: Error en el diseño previo	
Descripción	Un error de diseño obliga a retrasar o retroceder en la implementación.
Prevención	Realizar un diseñó adecuado y contrastado. Realizar el proyecto de manera modular, antes de pasar a la implementación.
Plan de contingencia	Si el diseño es modular, es posible encapsular los errores para no tener que realizar cambios totales al diseño general.
Probabilidad	Media.
Impacto	Media.

Tabla 2.8: 'Error de Diseño' de los Riesgos Técnicos

Implementación: Error en el implementación	
Descripción	Surgen errores de implementación imprevistos, los cuales conllevan a un retraso de la planificación y obligan a dedicar más tiempo del requerido inicialmente.
Prevención	Realizar un buen diseño previo ayudará a suplir este tipo de errores, así como una familiarización previa con las herramientas y lenguajes a utilizar. Prever el posible retraso dando holgura a las fechas previstas puede también de ayuda.
Plan de contingencia	Encapsular los errores surgidos, solucionarlos de manera individual para no generar más errores.
Probabilidad	Alta.
Impacto	Media.

Tabla 2.9: 'Error de Implementación' de los Riesgos Técnicos

Calidad: Fallos en el hardware	
Descripción	Durante el desarrollo suceden errores en el hardware, pueden ser por errores electrónicos o por mal funcionamientos. También pueden darse casos de virus informáticos
Prevención	Realizar mantenimientos y monitorización de los equipos de trabajo. Comprobar su estado de 'salud' cada cierto periodo.
Plan de contingencia	Disponer de un equipo de trabajo supletorio para no entorpecer el flujo de trabajo.
Probabilidad	Baja.
Impacto	Media.

Tabla 2.10: 'Fallos de Hardware' de los Riesgos Técnicos

Calidad: Aplicación defectuosa	
Descripción	La aplicación completa presenta errores de funcionamiento en la interfaz o está poco optimizada computacionalmente hablando.
Prevención	Diseñar los algoritmos y diagramas correspondientes de manera correcta desde un inicio.
Plan de contingencia	Comprobar posibles soluciones cambiando el enfoque de diseño de manera modular, realizar nuevas pruebas más exhaustivas.
Probabilidad	Baja.
Impacto	Alto.

Tabla 2.11: 'Aplicación defectuosa' de los Riesgos Técnicos

Riesgos de Gestión

Planificación: Incorrecta distribución de tareas	
Descripción	Se plantea incorrectamente la distribución de tareas en la fase de 'Planificación y gestión'. Ya que cierta tarea debería haber ido previa a otra tarea o viceversa. También puede suceder que la estimación temporal no sea la adecuada.
Prevención	Tener en cuenta en la planificación las entradas y salidas requeridas de cada fase. Distribuir un horario de trabajo eficiente desde un inicio, si es posible, para la correcta estimación temporal.
Plan de contingencia	Asignar nuevamente fechas de entrega, dependiendo de la tarea necesaria a realizar.
Probabilidad	Media.
Impacto	Media.

Tabla 2.12: 'Incorrecta distribución de tareas' de los Riesgos de Gestión

Seguimiento y Control: Falta de seguimiento	
Descripción	No se realiza de manera correcta o directamente de ninguna manera, un seguimiento previo a finalizar una tarea o alcanzar un hito.
Prevención	Concretar fechas de reunión con los supervisores y notificar debidamente los progresos conseguidos.
Plan de contingencia	Revisión de la tarea sin seguimiento, analizando los posibles problemas a ocasionar.
Probabilidad	Baja.
Impacto	Baja.

Tabla 2.13: 'Falta de seguimiento' de los Riesgos de Gestión

Planificación: Incumplimiento de fechas de entrega	
Descripción	Las tareas no se finalizan dentro de las fechas acordadas en la planificación.
Prevención	Acordar un calendario de entregas inicial con holgura para posibles retrasos.
Plan de contingencia	Reorganizar a corto plazo las tareas a entregar.
Probabilidad	Baja.
Impacto	Medio.

Tabla 2.14: 'Incumplimiento de fechas de entrega' de los Riesgos de Gestión

Riesgos Internos

Recursos humanos: Incapacidad de trabajo personal	
Descripción	Debido a problemas de salud, personales o familiares, la previsión de trabajo se ve retrasada o la calidad del mismo se ve mermada.
Prevención	Tener una fecha final con tiempo suficiente para poder recuperar trabajo. Disponer siempre de ayuda o supervisión para que la calidad y esfuerzo no disminuya. Realizar trabajo dentro de un horario para no hacer peligrar el proyecto.
Plan de contingencia	Recuperar horas de trabajo Pérdidas en horas fuera del horario previsto.
Probabilidad	Baja.
Impacto	Alta.

Tabla 2.15: 'Incapacidad de trabajo personal' de los Riesgos Internos

Prioridades: Priorización errónea de tareas	
Descripción	A la hora de realizar una tarea, se antepone otra a la prevista en el orden de tareas. Esto supone que pueda a perderse tiempo realizando actividades innecesarias.
Prevención	Organizar de manera adecuada desde un inicio, todos los paquetes de trabajo, objetivos y requisitos del proyecto.
Plan de contingencia	Realizar con premura la tarea sin priorizar.
Probabilidad	Baja.
Impacto	Media.

Tabla 2.16: 'Priorización errónea de tareas' de los Riesgos Internos

Riesgos Externos

Supervisión: Incapacidad de comunicación con los supervisores	
Descripción	Se pierde capacidad de supervisión y seguimiento con los supervisores. Lo que puede suponer problemas de incumplimiento de requisitos.
Prevención	Definir requisitos completos desde un inicio. Así como fechas de entrega.
Plan de contingencia	Comunicación por medios no establecidos con los supervisores.
Probabilidad	Muy baja.
Impacto	Muy Alto.

Tabla 2.17: 'Incapacidad de comunicación con los supervisores' de los Riesgos Externos

Supervisión: Insatisfacción del trabajo realizado	
Descripción	La falta de satisfacción por parte del supervisor, respecto a la aplicación y documentación realizada.
Prevención	Realizar a lo largo de todo el proyecto reuniones de seguimiento para así ir comprobando que se está realizando las tareas de acuerdo a lo establecido en la sección de requisitos.
Plan de contingencia	Revisión del trabajo realizado teniendo en cuenta las insatisfacciones.
Probabilidad	Media.
Impacto	Alto.

Tabla 2.18: 'Insatisfacción del trabajo realizado' de los Riesgos Externos

2.5. Evaluación económica

En el siguiente apartado se realizará la evaluación económica pertinente al proyecto. Justificando así la inversión a realizar si el proyecto fuese financiado y se tuviese que tener en cuenta todos los gastos económicos relativos. Para evaluarlo, el apartado se dividirá en los siguientes sub-apartados:

Mano de obra

El precio por hora de un programador informático promedio según el BOE [3] suele rondar los 10€ (1521,56€ mensuales / 152h de trabajo medio cada mes). Teniendo en cuenta que el proyecto se ha elaborado en 34 semanas, 4 de ellas siendo vacaciones, con un total de tres horas de trabajo medio diarias entre semana, dejando los miércoles libres, da como resultado 360 horas.

$$3 \text{ (horas) x } 4 \text{ (días) x } 30 \text{ (semanas)} = 360\text{h}$$

Teniendo en cuenta las horas invertidas y el salario medio, es necesario mencionar las retenciones del IRPF y Seguridad Social. Considerando el salario de un autónomo para este proyecto, se sabe que profesionales autónomos deben aplicar un IRPF del 15 % a los ingresos que obtengan por su actividad [4]. Por tanto en la Tabla 2.19 es posible visualizar el resultado.

Salario Neto	360 horas * 10 €/hora	3600 €
IRPF	3600 · 0,15	540 €
Seguridad Social	3600 · 0.10	360 €
Salario Total	3520 - 540 - 360	2640 €

Tabla 2.19: Coste de la mano de obra

Software y Hardware

El ser un proyecto orientado al desarrollo de un software, en este caso un compilador, se debe tener en cuenta el gasto de las herramientas/programas utilizadas y el hardware empleado para manejarlas.

El equipo utilizado es un Xiaomi Air 13.3[1] valorado en 900€, un ratón Logitech M330 valorado en 30€, un teclado Anne Pro 2 valorado en 80€ y para el desarrollo e investigación fuera del lugar de trabajo un Xiaomi Redmi Note 4X valorado en 180€. A este último hay que sumar un accidente que requirió el cambio de pantalla con un coste de 50€.

[1] https://www.pccomponentes.com/xiaomi-mi-air-133

Dispositivos	Coste monetario
Ordenador portátil	900 € / 8 meses de uso
Ratón	30 €
Teclado	80 €
Teléfono móvil	180 € + 50 €
Coste Total	1240 €

Tabla 2.20: Costes del Hardware

Sabiendo el costo total del hardware, es necesario calcular la amortización para determinar el valor real de estas herramientas (Tabla 2.21). Hay que tener en cuenta que el tiempo de vida de un ordenador portátil es de 4 años, de un móvil 2 años, de un ratón 2 años, y de un teclado mecánico 6 años. La estimación temporal de este proyecto han sido 32 semanas o lo que es lo mismo, 8 meses.

Dispositivos	Estimación temporal	Amortización
Ordenador portátil	$\frac{900/4 \cdot 8}{12}$	150 €
Ratón	$\frac{30/2 \cdot 8}{12}$	10 €
Teclado	$\frac{80/6 \cdot 8}{12}$	8,89 €
Teléfono móvil	$\frac{230/2 \cdot 8}{12}$	76,67 €
Amortización total		245,56 €

Tabla 2.21: Costes de amortización

En este proyecto no se han hecho gastos de software. La totalidad de los programas utilizados son gratuitos. Si no lo fueran se han usado versiones de prueba o licencias cedidas por los supervisores.

Gastos varios

Otros tipos de gastos son necesarios a tener en cuenta. Gastos relativos al consumo eléctrico, de internet, desplazamientos, datos móviles etc.

Gastos varios	Estimación temporal	Gasto
Luz	$40 \cdot 8$	320 €
Internet	$38 \cdot 8$	304 €
Desplazamiento	$\frac{260 \cdot 8}{12}$	174 €
Datos móviles	$12 \cdot 8$	96 €
Gasto total		894 €

Tabla 2.22: Gastos varios

Coste total del proyecto

Finalmente, resta evaluar el coste total del proyecto incluyendo todos los apartados anteriores.

Salario	2700 €
Gastos de amortización	245,56 €
Gastos de Software	0 €
Otros gastos	894 €
Coste total del proyecto	3.839,56 €

Tabla 2.23: Coste total del proyecto

3.779,56 € sería el coste del proyecto completo. Se estima que la ganancia personal sería superior al coste total del proyecto.

CAPÍTULO 3

Captura de Requisitos

Para que el proyecto llegue a buen puerto, es necesaria una correcta captura de requisitos. Sin ella, es difícil saber dirigir el trabajo a realizar, ya que no se tiene un objetivo u objetivos concretos.

3.1. Objetivos

Una vez se tuvo en cuenta el tema y dirección del proyecto, se quiso establecer el alcance del mismo. Definiendo así unos objetivos, tanto principales como secundarios.

Principales

Estos son los objetivos principales a completar. Lo mínimo requerido para que el proyecto se considere completo.

- Definir un lenguaje intermedio para la escritura de acordeón
- Crear un compilador para convertir el lenguaje intermedio a Lilypond, desde la cuál se obtendrá el archivo PDF. El compilador tendrá en cuenta las siguientes notaciones:
    - Melodía en clave de Sol
    - Bajo en clave de Fa
    - Notas y Acordes de distinta altura
    - Alteraciones
    - Digitación y escritura numérica de acordeón
- Mostrar la escritura de trikitixa junto a la partitura compilada
- Página web donde poder realizar las siguientes funciones:
    - Importar un archivo de texto en lenguaje intermedio y exportarlo a PDF

- Escribir en la propia web el lenguaje intermedio para poder compilar y obtener una previsualización

Secundarios

Los objetivos secundarios han sido pensados como añadidos al proyecto general. Estas serán funcionalidades extra que no comprometan al trabajo si alguno falta.

- Implementar la exportación de lenguaje intermedio a MIDI
- Mostrar el resultado en MIDI en la web al compilar
- Añadir siguientes notaciones al lenguaje intermedio y por tanto a la partitura:
  - Silencios
  - Marcas de compás y repeticiones
  - Definición de compás
  - Notas con medidas y valores irregulares
  - Notas con puntillo
  - Indicadores de expresión (ligaduras, *staccato*...)
  - Intensidad (forte, piano...)
  - Tempo (*adagio*, *allegro*...)
- Implementar el cifrado americano para los acordes según cambian en la partitura
- Seguimiento de la partitura mientras se reproduce el MIDI en la web

3.2. Análisis de antecedentes

A la hora de seleccionar herramientas para la elaboración del proyecto, se barajaron distintas alternativas.

Alternativas actuales

Para poder crear una aplicación con un objetivo claro y cumpliendo los requisitos acordados, se ha tenido que realizar una investigación previa. El objetivo era encontrar, analizar y contrastar herramientas de funciones similares para así saber cual sería la adecuada para el proyecto. Algunas herramientas servirían como inspiración y otras se utilizarían en el propio proyecto.

Todo empieza en la web Trikitirauki[1], donde es posible visualizar, crear y compartir partituras de trikitixa mediante una interfaz sencilla a la vez que intuitiva para los trikitilaris. Indagando, se ha comprobado que no existen alternativas para poder pasar de esta manera de escritura a una partitura convencional, al no ser este tipo de escritura algo extendido fuera del País Vasco. Es un

[1] https://tirikitrauki.com/zenbakiz/

estándar muy habitual en la enseñanza del acordeón vasco, pero al estar muy concentrado en este territorio no ha habido muchas propuestas de soluciones tecnológicas al respecto.

Estudio de posibles alternativas

Por tanto el objetivo era conocer posibles herramientas con las que poder enlazar este tipo de escritura particular con un pentagrama universal.

Edición de partituras

Se barajaron diversas alternativas de edición de partituras de pentagrama clásico por si alguna ofrecía algún tipo de herramienta con la que poder trabajar de esta manera.

Dorico: Es un software de escritura de partituras. Junto con Finale y Sibelius, es uno de los tres principales programas de notación musical a nivel profesional.

Finale: Creado por MakeMusic, es probablemente el programa de edición musical más extendido y utilizado. Existen muchas variaciones del mismo para distintos ámbitos.

Musescore: Uno de los programas de notación musical más usados al ser de software libre y código abierto. Contiene una de las interfaces gráficas más completas y gran variedad de formatos para exportación e importación. Pudiendo transformar partituras en cierta codificación en MIDI por ejemplo.

ScoreCloud ScoreCloud 4 (antes 'ScoreCloud Studio') es un software de escritorio para Windows y Mac OSX que crea notación musical a partir de lo que tocas. También posee una interfaz de edición de partituras y sonido.

Sibelius: Podría ser el 'Office' de la edición y escritura de partituras musicales por su interfaz e intuitividad. Similar en funcionalidad a Musescore.

Aún así ninguna de las herramientas estudiadas presentaba algún mínimo acercamiento a ofrecer una posible solución con la que partir donde empezar a desarrollar el proyecto. Se propuso entonces la idea de crear un traductor donde mediante un lenguaje amigable y en texto plano totalmente legible por un trikitilari, se obtuviese su equivalente en partitura. Por ello, el enfoque cambió y se empezó a buscar tipos de notación musical en texto plano, con las que por texto se pudiese obtener de manera totalmente personalizada el resultado requerido.

Representación y notación musical

Entre diversas webs y plataformas, se tuvo en consideración el trabajo *XMLScore*, disponible a través de la plataforma ADDI de la UPV/EHU [5]. Proyecto realizado por el ya graduado alumno Aitor Valle, donde se barajaron alternativas que pudieron ser de utilidad para este proyecto. Estas

fueron algunas de las barajadas:

JFugue: Es una biblioteca de programación de código abierto para Java, con la que es posible programar música alejándose de las complejidades del MIDI.

JMusic: Es un proyecto diseñado para proporcionar a los compositores y desarrolladores de software una librería de herramientas de composición y procesamiento de audio. De código abierto y escrito en Java.

Lilypond: Utilizando una sencilla notación de texto plano como entrada, se presenta como alternativa de software libre para la creación y edición de partituras.

MusicXML: Diseñado para el intercambio de partituras, especialmente entre distintos editores como los vistos previamente. MusicXML es un formato abierto, basado en XML, de notación musical.

Al final se optó por **Lilypond**, la opción más completa y flexible donde se podría generar la partitura necesaria de la manera que se requiriese. Con Lilypond no se necesitaría ningún lenguaje aparte como Java, o editor como Musescore. Además es posible obtener el resultado de la partitura en PDF.

Compiladores

Ahora quedaba decidir la herramienta con la que se traduciría el lenguaje intermedio creado (Capitulo 5.1) a notación Lilypond. Con la intención de crear un compilador se pusieron sobre la mesa varias alternativas:

ANTLR: Es una herramienta para la creación de analizadores sintácticos (parsers), intérpretes, compiladores y traductores de lenguajes, partiendo de la gramática descrita por los mismos. Utiliza algoritmos 'LL' de parsing partiendo de la descripción formal de la gramática de un lenguaje.

Lex/Flex - Yacc/Bison: Lex es un programa informático que genera analizadores léxicos. Flex es la alternativa más completa de Lex. Ambos pueden ir vinculados a Yacc/Bison. Donde Yacc es un programa de generación de analizadores sintácticos o parsers. Bison es compatible completamente con Yacc y posee extensiones que este último no contiene. Bison genera por defecto analizadores LALR(1), pero también puede generar analizadores canónicos LR, IELR(1) y GLR.

Ocaml: Es un lenguaje de programación multiparadigma de propósito general. Amplía el dialecto Caml de ML (Meta Language) con características orientadas a objetos. Incluye un intérprete interactivo de nivel superior, un compilador de código de bytes, un compilador de código nativo optimizado y un depurador reversible.

Al final se terminó optando por **Flex y Bison** como base del compilador. Lex y Yacc también requerirán atención, ya que su funcionamiento con sus versiones actuales es prácticamente la misma. Esta decisión se llevó a cabo, entre otras razones, ya que estas herramientas permiten diseñar de manera totalmente personalizada el compilador, pudiendo definir en este caso el lenguaje intermedio de la manera requerida. A su vez, son la opción que más curva de aprendizaje y enseñanzas van a ofrecer respecto a el mundo de los compiladores. En el Capítulo 5.3 se añaden más razones de esta elección tomada.

Desarrollo Web

Finalmente solo queda elegir una plataforma donde crear la web y poder vincular el compilador con ella. Así se ofrecerá esta herramienta a todo el que lo necesite y disponga de conexión a una red de Internet. Estas fueron algunas de las opciones a escoger:

ASP.NET: Desarrollado por Microsoft, es un entorno para el desarrollo de sitios web dinámicos y aplicaciones web.

Django: Desarrollado en Python y dirigido al desarrollo web escalable, es un potente framework que respeta el patrón MVC (Modelo Vista Controlador).

Flask: Es un micro-framwork para el desarrollo de API's con gran carga de visitas. Es utilizado para proyectos personalizados y de gran sencillez.

Ruby on Rails: Escrito en el lenguaje de programación Ruby y siguiendo el patrón MVC, es un framework de aplicaciones web de código abierto.

El proyecto se acabó desarrollando en **Django**. Entre otras cosas, por cumplir con el patrón MVT (Model-View-Template, una variación del patrón de arquitectura MVC) el cual simplifica y aclara la relación con la herramienta, por contar con una estructura del proyecto auto-generado y por ser más escalable y capaz que el resto. Como lo demuestran grandes empresas como Instagram, Mozila, The Washington Times, Disqus, Bitbucket y Nextdoor.

Inspiraciones

Para poder enfocar correctamente la aplicación y tener un rumbo claro, se tuvieron en cuenta ciertas inspiraciones. Estas se intentaron tener en cuenta lo máximo posible, para poder asemejar sus puntos positivos, a objetivos a alcanzar con nuestra aplicación. Entre otras, las principales fueron:

Overleaf: Esta fue la primera inspiración visual de la herramienta. Al querer desarrollar un compilador online con una entrada de texto, con el input necesario y un botón donde pulsar para ver de manera contigua el resultado (Figura 3.1). Pudiendo compilar en este caso

el input en formato LaTeXy obtener un PDF como resultado.

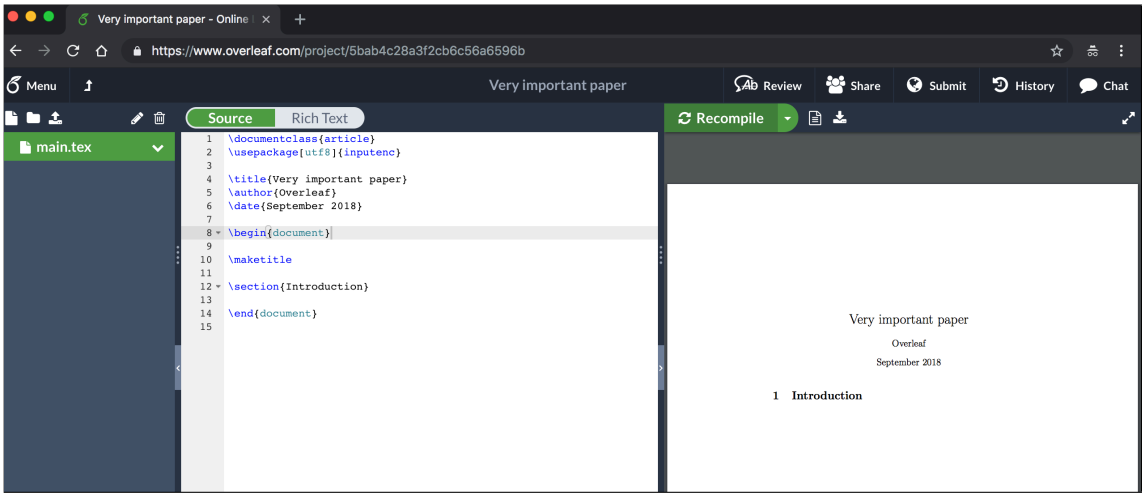


Figura 3.1: Ejemplo de Overleaf

Hacklily: Esta web donde se pueden realizar inputs en Lilypond y de una manera clara comprobar sus errores y el resultado contiguo fue aún más una inspiración clara para el proyecto. Además de ser software libre y parte del proyecto GNU (Figura 3.2).

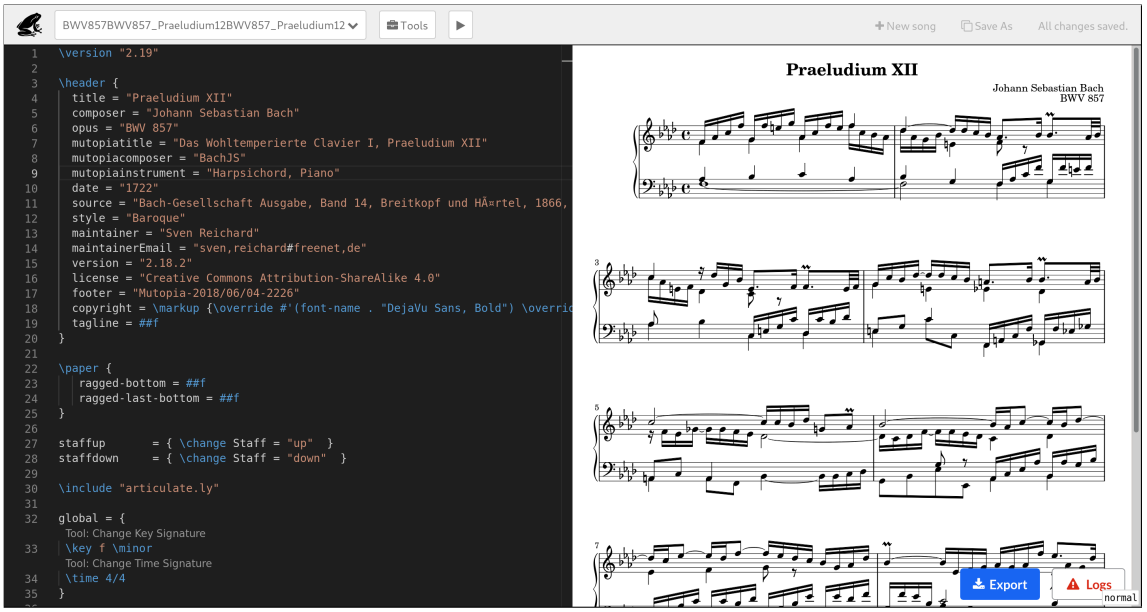


Figura 3.2: Ejemplo de Hacklily

Frescobaldi: Aunque sea un descubrimiento de último momento e incluye funciones que no se van a suplir, como la edición de partitura tanto con Lilypond como con la interfaz, es una herramienta a tener en cuenta (Figura 3.3).

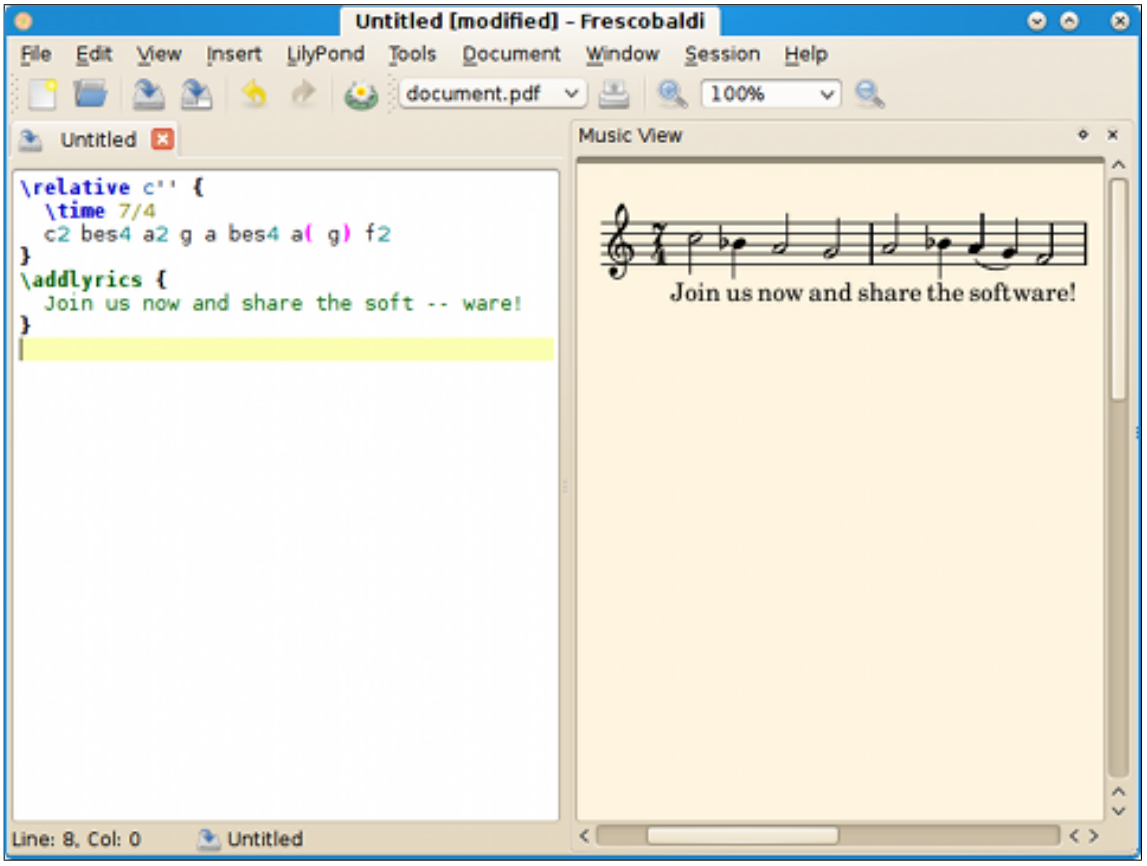


Figura 3.3: Ejemplo de Frescobaldi

CAPÍTULO 4

Análisis y diseño

Para plantear el desarrollo del trabajo y en qué partes se divide el mismo se ha realizado los correspondientes diagramas para saber identificarlas.

4.1. Casos de uso

Para poder identificar la utilidad de la aplicación, es necesario realizar los casos de usos correspondientes con los actores y usos necesarios.

Compilación

En la aplicación aquí implementada solamente tendrá acceso un tipo de usuario, que será cualquiera que requiera pasar de una partitura de trikitixa a una partitura de pentagrama.

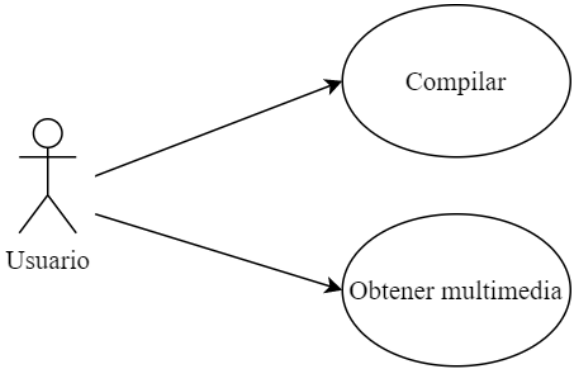


Figura 4.1: Caso de Uso: Compilar

Compilar

Descripción

Permite al usuario introducir y después compilar el lenguaje intermedio en una partitura. El resultado será la partitura compilada y un mensaje de éxito o un mensaje de error.

Actor

- Usuario

Precondiciones

Ninguna.

Requisitos no funcionales

Ninguno.

Flujo de eventos

1. El usuario pulsa el botón 'Compilar'.
   [Si ha introducido previamente una partitura sin errores o vacía]
   a) Se muestra la partitura correspondiente y un mensaje de éxito
   [Si ha introducido previamente una partitura con errores]
   a) No se muestra la nueva partitura compilada, indicando la localización del el primer error encontrado

Obtener multimedia

Descripción

El usuario podrá solicitar archivos multimedia correspondientes a la partitura (PDF, PNG, MIDI), en caso de no haber compilado previamente, se mostrará un mensaje de aviso.

Actor

- Usuario

Precondiciones

Ninguna.

Requisitos no funcionales

Ninguno.

Flujo de eventos

1. El usuario pulsa cualquier botón correspondiente a la multimedia.
   [Si ha introducido previamente una partitura correcta]
   a) Se muestra la partitura en el formato solicitado.
   [Si no se ha introducido previamente ninguna partitura]
   a) No muestra nada y avisa de la necesidad de compilar previamente la partitura.

4.2. Patrón MVT

El diseño de la aplicación está orientado al patrón MVT (Model Template View), ya que es el patrón que utiliza Django para su funcionamiento y despliegue. Este patrón es similar al visto durante el grado llamado MVC (Modelo Vista Controlador) pero con ciertas diferencias (Tabla 4.1).

MVC	**MVT**
Model: Esta capa se ocupa de la lógica relacionada con los datos. Puede recuperar, modificar y guardar datos en la base de datos.	**Model**: Igual que en el patrón MVC, en este caso con los ficheros usados y generados por el compilador. En Django se identifica con el fichero `models.py`.
View: Se encarga de recoger los datos del modelo o del usuario y de presentarlos. En una aplicación web, todo lo que se muestra en el navegador cae bajo esta capa de presentación.	**View**: En el patrón de diseño MVT, la vista decide qué datos deben mostrarse. Se identifica con el fichero `views.py`.
Controller: Controla el flujo de datos y la interacción entre la vista y el modelo. Por ejemplo, un controlador, basado en una solicitud o acción, recogerá datos de una base de datos con la ayuda del Modelo y los enviará al usuario a través de las Vistas.	**Template**: Las plantillas se utilizan para especificar una estructura para una salida. Los datos pueden rellenarse en una plantilla utilizando marcadores de posición. Define cómo se presentan los datos. En Django se identifica con ficheros `.html` dentro del directorio `templates`.

Tabla 4.1: Comparación entre los patrones MVC y MVT

Como explicación general, el patrón MVT comienza con el acceso del usuario a la URL a través del navegador. Su acceso es tratado por el mapeo de URLs que le muestra la plantilla correspondiente. Cuando el usuario interactúa con la plantilla, el input generado es gestionado por la vista, la cual

comprueba si fuese necesario la validez de las modificaciones (forms). Si es necesario crea, edita y ofrece datos correspondientes a la lógica de la aplicación, pasará a models los detalles de los datos a tratar. Este último interactúa con la base de datos correspondiente para acceder o modificar los datos necesarios. La representación gráfica de este proceso puede verse en la Figura 4.2.

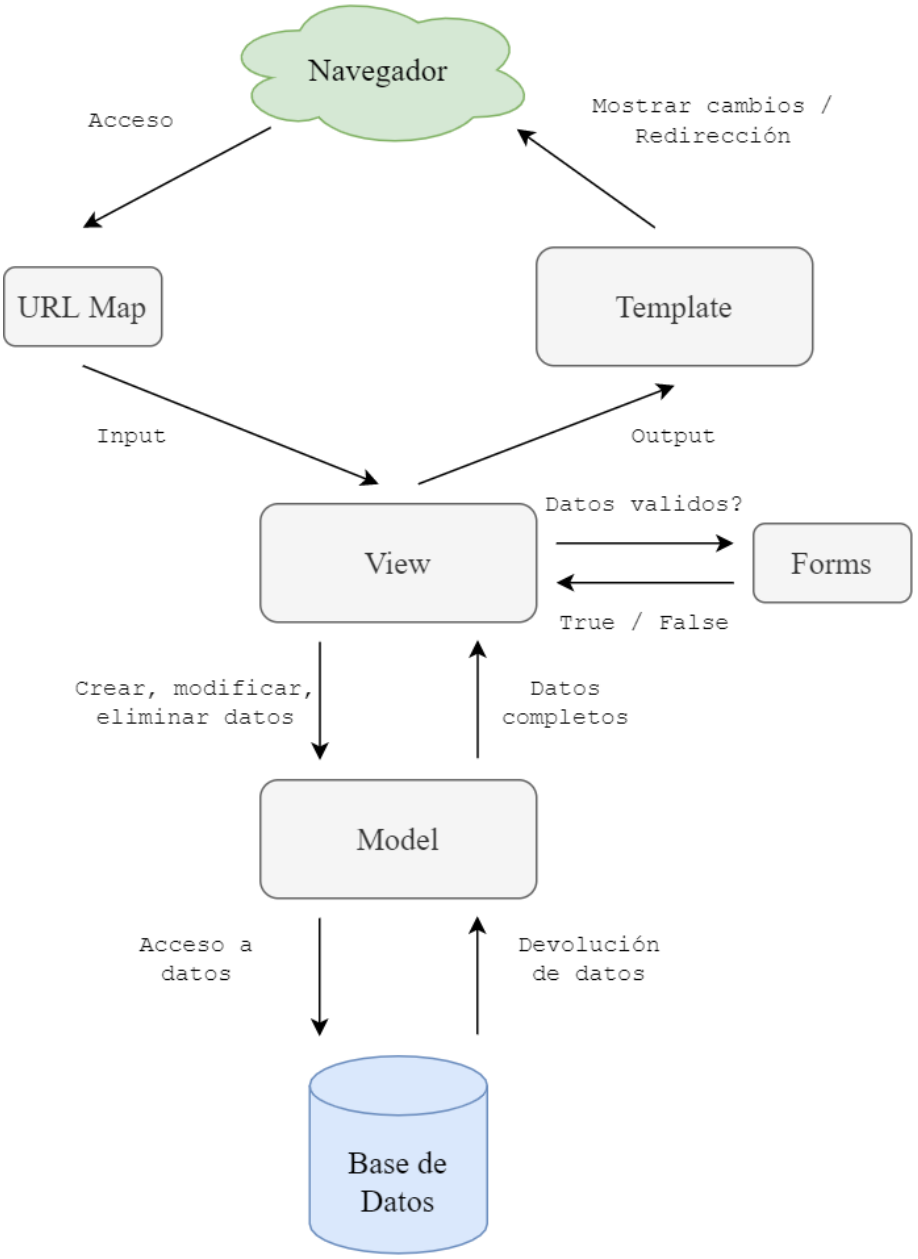


Figura 4.2: Patrón MVT

En el caso de este proyecto, el proceso sería el siguiente. El usuario accedería a través del navegador a la URL correspondiente a la aplicación. El fichero urls.py sería el encargado de la redirección seleccionando la ruta `compilador/`, que es quien decide qué vista (views.py) debe

ejecutarse, siendo la vista la que decide la plantilla. Aquí es donde el usuario introducirá su lenguaje intermedio y pulsaría el botón correspondiente a realizar la compilación. Esta información pasaría a ser gestionada por views.py donde se aloja la vista `compilador`, quien tiene acceso al fichero models.py donde se almacena la clase `compilador`. Esta clase se encarga de la interacción con las salidas y entradas del compilador, gestionando y ejecutando los comandos correspondientes. Se devolverán los archivos correspondientes (PDF, PNG, MIDI) en cada caso. Estos ficheros son transferidos a la vista y comprobados por ella para que el template pueda mostrarlos, reflejándose este cambio en el navegador del usuario. La representación gráfica de este proceso puede verse en la Figura 4.3.

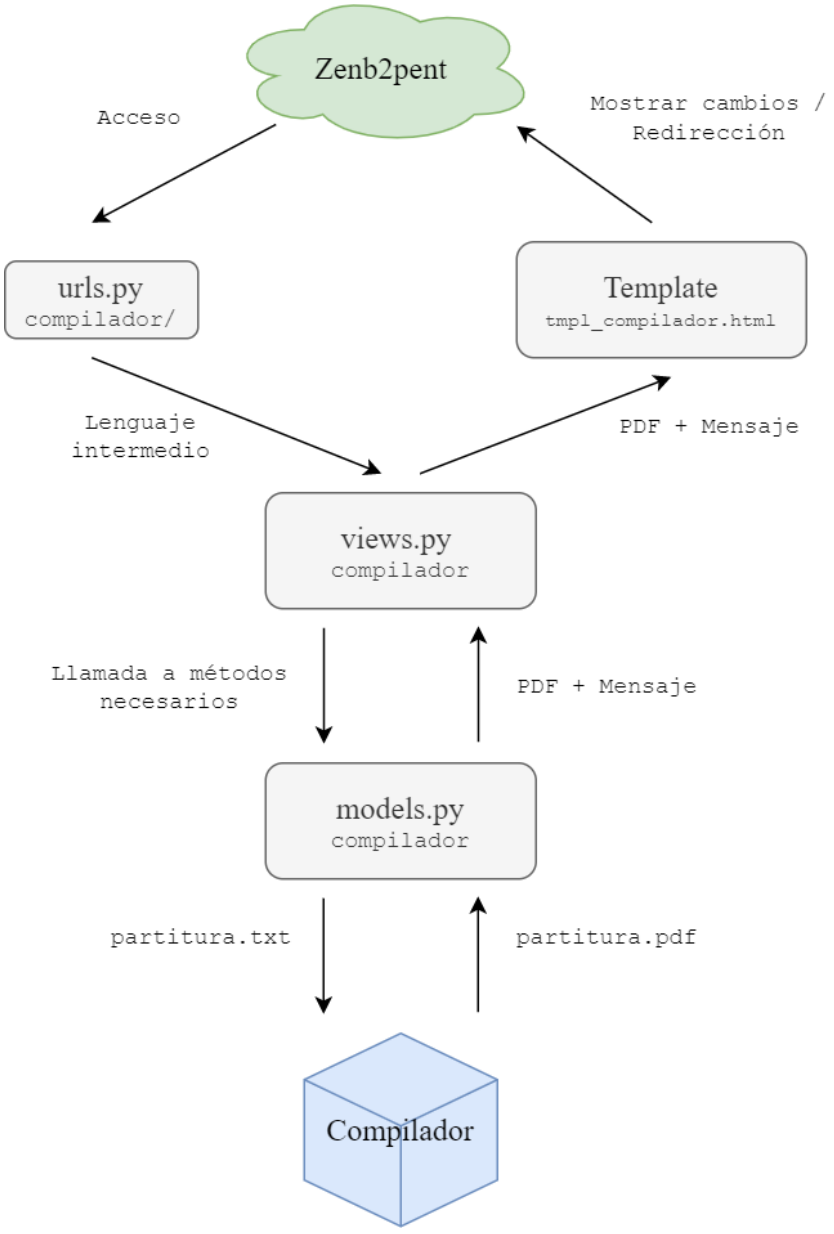


Figura 4.3: Patrón MVT

CAPÍTULO 5

COMPILADOR

En el siguiente apartado se especificará la implementación del compilador. Inicialmente se explicará la gramática y léxico del «lenguaje intermedio» creado para representar el lenguaje "zenbakizko". Seguidamente el resultado que se obtendrá al pasar de ese lenguaje a la notación utilizada por Lilypond. Después se especificará el funcionamiento del compilador en este caso concreto. Se finalizará con un análisis de las herramientas Flex y Bison, su uso y especificación. Con esto se quiere llevar a cabo un acercamiento a estas utilidades, y así facilitar al usuario el uso de estas herramientas.

5.1. Lenguaje intermedio

A fin de que el compilador cumpliese con los requisitos previstos, se ha realizado como primer paso, un análisis de las partituras de trikitixa para crear un lenguaje intermedio legible y comprensible por cualquier trikitilari. Si a lo largo de la sección surgieran dudas sobre las características del lenguaje y de su forma de escribir música, es posible consultar el Apéndice A. Asimismo se ha puesto como requerimiento, que tenga una curva de aprendizaje baja, para así facilitar el uso de la herramienta a cualquier músico de está especialidad.

Para crear una gramática amigable para este lenguaje intermedio, se ha decidido definir un lenguaje libre de contexto (LLC). Estos lenguajes forman una parte más amplia de los lenguajes regulares (Figura 5.1). Sus diferencias serían:

- Los lenguajes regulares pueden ser representados por una gramática regular y por un autómata finito determinista[1]. Estos lenguajes pueden ser comprensibles dentro de su léxico leyéndolos de izquierda a derecha y viceversa.
- Los lenguajes libres de contexto son representados por gramáticas libres de contexto y por autómatas de pila. Estos lenguajes son comprensibles únicamente en una dirección [6].

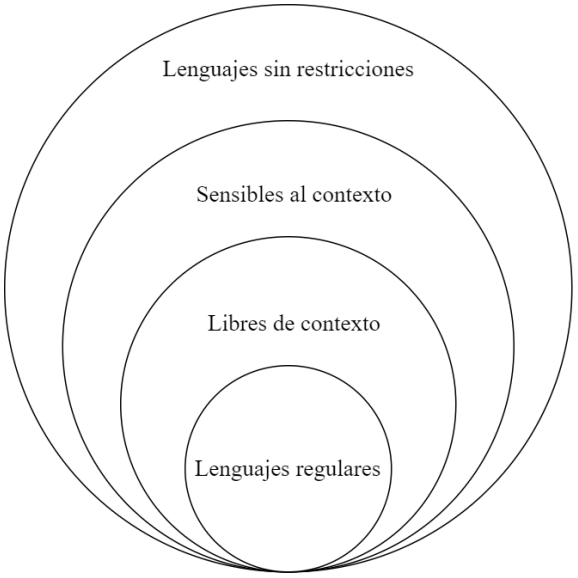


Figura 5.1: Tipos de lenguajes

Para definir el LLC se ha creado una gramática que no presenta ambigüedades, lo cual es indispensable para crear analizadores sintácticos eficientes. Es decir, cualquiera de las sentencias que se escriban como entrada del compilador con este lenguaje intermedio tendrá solamente una manera de generarse en la gramática libre de contexto presentada en este trabajo (véase Apéndice C). Siempre y cuando la sentencia cumpla con las reglas gramaticales y esté dentro del léxico definido. Dicho de otra manera, cualquier frase del lenguaje tiene una sola derivación en la mencionada gramática.

Notación

La trikitixa cuenta con la posibilidad de reproducir varios tipos de sonidos al mismo tiempo, pudiendo combinar melodía y bajo. A su vez, cuenta con dos distribuciones de botones, uno al lado derecho para la melodía con 23 botones y otro para el izquierdo para el bajo con 12 botones (ver Figura 5.2). Todo el apartado de las notas se basa en esta distribución y en la manera de reproducir la música que tiene la trikitixa. A continuación se indicará la representación individual de estas voces.

[1] https://www.institucional.frc.utn.edu.ar/sistemas/ghd/T-M-AFD.htm

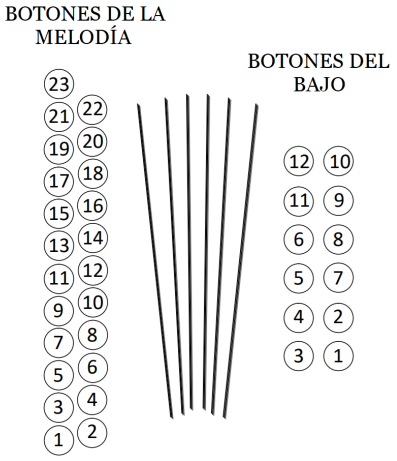


Figura 5.2: Botones de la trikitixa

Melodía

La melodía es una sucesión de sonidos con sentido musical, la cual se reproduce de manera ordenada. En el caso concreto del acordeón, su sonido no va ligado únicamente al botón del margen derecho que se pulse, sino también al estado del fuelle. Se puede ver por la escritura propia de la trikitixa, que una nota de la melodía puede por tanto tener el mismo número pero dos significados distintos siendo causa de esto el estado del fuelle.

Disponiendo de 23 botones, sabiendo que el registro melódico del acordeón es de «Do # 4 - Re 6» cuando el fuelle está abriendo y «Mi b 4 - Sol 6» cuando cierra. El lenguaje entonces tiene que ser capaz de representar cada una de estas notas de manera escrita, diferenciando en todo momento el estado del fuelle.

Para representar un ejemplo se tomará el botón '11', el cual representa «Si-5» cuando el fuelle abre y «Do-5» mientras cierra. En la escritura propia de la trikitixa se representa con una circunferencia fina cuando el fuelle cierra y una gruesa cuando abre (Figura 5.3).

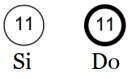


Figura 5.3: Botón Nº 11 de la melodía

Es importante definir en nuestro lenguaje la situación del fuelle. Por ello se ha decidido representar una nota de la melodía de la siguiente manera (Figura 5.4).

Figura 5.4: Ejemplo del «lenguaje intermedio»

Se ve que la representación de la escritura para las dos notaciones, es similar. Siendo 'c' y 'a' los indicadores del estado del fuelle, 'cerrando' y 'abriendo' respectivamente. Después, con los paréntesis se indicará qué botones de la trikitixa se van a tocar. La representación completa de los botones en parte de la melodía de la trikitixa es posible verla en la Tabla 5.1.

	Trikitixa		**Lenguaje intermedio**	
Botón	Cerrando	Abriendo	Cerrando	Abriendo
1	①	①	c(1)	a(1)
2	②	②	c(2)	a(2)
3	③	③	c(3)	a(3)
4	④	④	c(4)	a(4)
5	⑤	⑤	c(5)	a(5)
6	⑥	⑥	c(6)	a(6)
7	⑦	⑦	c(7)	a(7)
8	⑧	⑧	c(8)	a(8)
9	⑨	⑨	c(9)	a(9)
10	⑩	⑩	c(10)	a(10)
11	⑪	⑪	c(11)	a(11)
12	⑫	⑫	c(12)	a(12)
13	⑬	⑬	c(13)	a(13)
14	⑭	⑭	c(14)	a(14)
15	⑮	⑮	c(15)	a(15)
16	⑯	⑯	c(16)	a(16)
17	⑰	⑰	c(17)	a(17)
18	⑱	⑱	c(18)	a(18)
19	⑲	⑲	c(19)	a(19)
20	⑳	⑳	c(20)	a(20)
21	㉑	㉑	c(21)	a(21)
22	㉒	㉒	c(22)	a(22)
23	㉓	㉓	c(23)	a(23)

Tabla 5.1: Representación de los botones de la melodía en el lenguaje intermedio

Aún falta la representación de diversos sonidos melódicos sonando al mismo tiempo. Ya que, en la escritura de acordeón se pueden representar de manera escrita dichas representaciones. La diferencia con las ya mencionadas viene principalmente a que van pegadas y agrupadas en grupos de dos, tres o cuatro. Pudiendo ser tanto con el fuelle abriéndose (Figura 5.5):

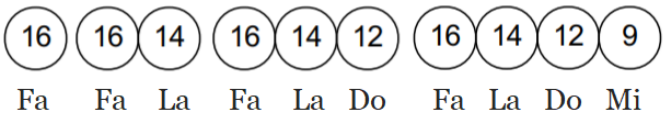


Figura 5.5: Múltiples botones con el fuelle abierto

Como cerrándose (Figura 5.6):

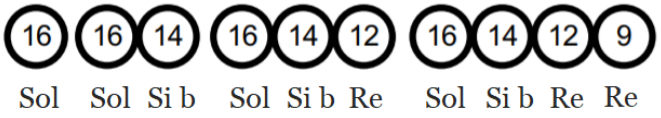


Figura 5.6: Múltiples botones con el fuelle cerrado

Es posible entonces, hacer una amplia gama de combinaciones a la hora de combinar notas de la melodía. Pero siempre serán un máximo de 4, ya que es lo que un trikitilari corriente puede tocar con una sola mano.

Para solucionar esto, se ha decidido añadir, dentro del paréntesis de la melodía las nuevas pulsaciones. Separándolas siempre por comas, para que el compilador sea capaz de distinguir un número de otro (Tabla 5.2).

Trikitixa	Lenguaje intermedio
16	c(16)
16 14	c(16,14)
16 14 12	c(16,14,12)
16 14 12 9	c(16,14,12,9)
16	a(16)
16 14	a(16,14)
16 14 12	a(16,14,12)
16 14 12 9	a(16,14,12,9)

Tabla 5.2: Representación de múltiples botones simultáneos de la melodía en el lenguaje intermedio

Teniendo esto como ejemplo, ya es posible representar cualquiera de los sonidos referentes a la melodía del acordeón en nuestro lenguaje. Pero aún faltaría la digitación de los mismos.

La digitación del acordeón es algo especial, ya que esta basado en símbolos y no en números como suele ser habitual. Se emplean para ello todos los dedos de la mano derecha, menos el pulgar (Figura 5.7). Como es evidente, cada nota tendrá únicamente una digitación al mismo tiempo.

Figura 5.7: Digitación del acordeón

- ‘ ’ si la notación es vacía, se debe usar el dedo índice
- • si es un circulo negro, se usa el corazón
- ∘ con un circulo blanco, se usa el anular
- **x** con la cruz se debe usar el meñique

En una serie de notas de melodía el resultado sería el que se muestra en la Figura 5.8:

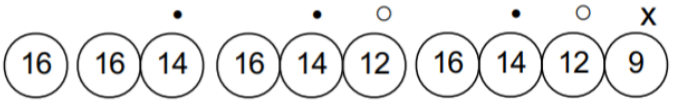


Figura 5.8: Digitación en notas de la melodía

Se ha tenido que tener en cuenta la existencia de una digitación en la escritura de la trikitixa. Por lo que para representarlo en nuestro lenguaje con cierta lógica y para que sea intuitivo respecto al lenguaje de trikitixa se ha decidido lo siguiente:

- Si se quiere representar al dedo índice se pondrá ‘v’ de vacío, o nada junto al número.
- Con ánimo de representar al dedo corazón se pondrá ‘n’ del círculo negro.
- Para representar al dedo anular se pondrá ‘b’ del círculo blanco.
- Si se quiere representar al dedo meñique ‘x’ de la propia cruz.

Como último punto, cada vez que se finalice una línea del lenguaje intermedio, en otras palabras, se acabe una nota de la melodía, se finalizará con un ';' para que el compilador reconozca el fin de ese sonido y el comienzo de otro. Teniendo esto en cuenta en la Figura 5.9 se muestran algunos ejemplos con la notación de la melodía completa.

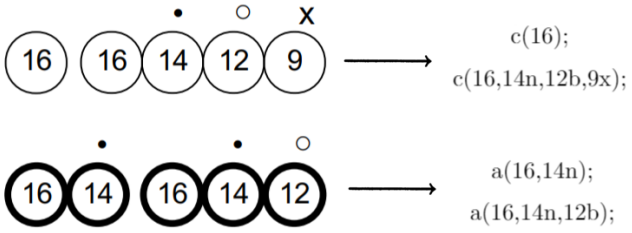


Figura 5.9: Sección de melodía, comparación de acordeón a lenguaje

Las decisiones tomadas para representar la melodía en el lenguaje intermedio son las siguientes:

- Cada conjunto de sonidos simultáneos en la melodía se indicará en una línea.
- Se requerirá indicar siempre el estado del fuelle, previo a indicar los botones a pulsar.
- Siempre se deberá abrir y cerrar el paréntesis con el contenido de la melodía en el interior.
- Si la melodía cuenta con más de un sonido, estos se separarán por comas.
- El ';' vendrá a continuación del paréntesis de cierre de la melodía.

Bajo

Al revisar suficientes partituras de trikitixa, se sabe que la melodía suele ir mayoritariamente acompañada de un bajo. Como es de suponer, existen reglas propias dentro del bajo, por lo que a continuación se dará una explicación de cómo se han tratado en nuestro lenguaje intermedio.

Los números del teclado izquierdo correspondientes al bajo, suelen indicarse justo debajo de los botones correspondientes del teclado derecho, es decir la melodía (Figura 5.10). Es posible tocar un gran número de sonidos del bajo junto a la melodía, ya que salvo en combinaciones especiales que más adelante se especificarán, estos se tocan consecutivamente. En el ejemplo de la Figura 5.10 se pueden ver distintas maneras de representar los botones del bajo: separados por comas o simplemente uno detrás de otro.

Cuando los números del bajo los separan por comas, quiere indicar que por cada coma, las notas de la melodía que corresponden a ese bajo vuelven a tocarse. Es decir, que en el ejemplo previo, el conjunto con el bajo `7,8,7,8`, indica que se tocará de nuevo la melodía, con cada número del bajo que hay previo a una coma. Esto habrá que tenerlo en cuenta en el compilador, ya que significará

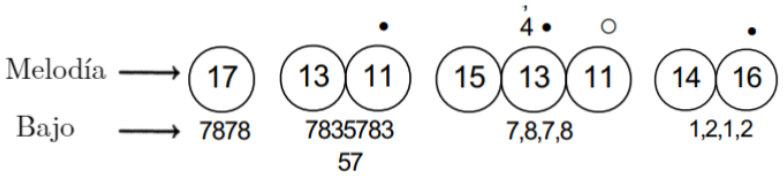


Figura 5.10: Sección con bajo, ejemplo corriente

un nuevo compás a generar. En algunos casos se indica encima de las notas de la mano derecha el número de veces que deberán pulsarse esos botones (primer caso de la Figura 5.11).

En vez de las comas, pueden estar simplemente separados por espacios o juntos unos de otros. Ambos significan lo mismo, que estas notas del bajo se tocan mientras dura la melodía. Puede darse que el bajo esté en dos líneas distintas, no significando esto nada en especial y siendo su función la misma que ponerlo en línea. Esto se da ya que estas partituras suelen escribirse a mano, y el espaciado no es muy meticuloso.

Todas las opciones anteriormente mencionadas pueden combinarse entre sí, por tanto se ha decidido implementar el bajo como se puede ver en la Figura 5.11.

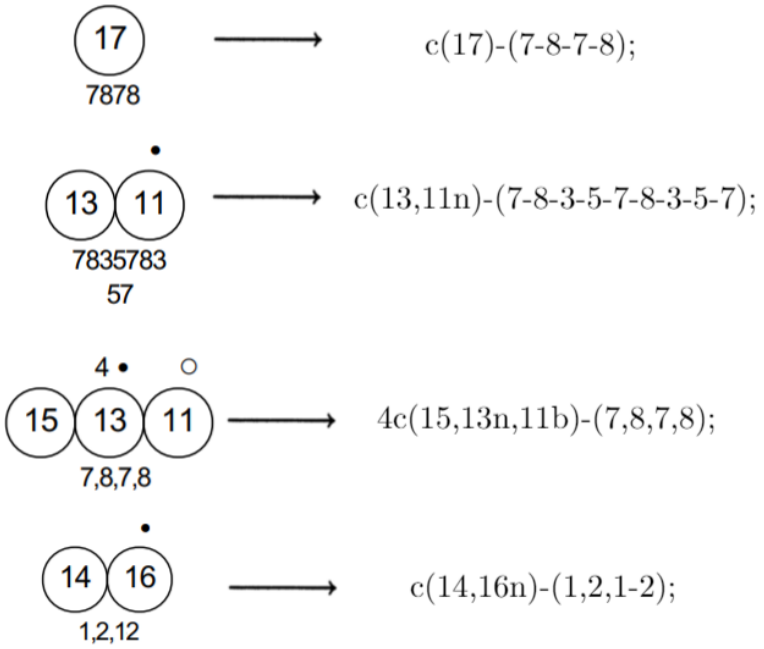


Figura 5.11: Representación del bajo

En la Figura 5.11 se muestra la representación de las notas del bajo en este lenguaje intermedio. Estas notas irán entre paréntesis precedidos de un guión, este guión sirve para separar la melodía

del bajo. Cuando en la parte superior del lenguaje de trikitixa aparece un número, este se pondrá justo al principio de la línea, indicando cuántas veces se deberá repetir la melodía con su bajo correspondiente.

Los números correspondientes a los botones estarán separados por comas si así están representados en el lenguaje de acordeón, y por guiones para la representación por espacios.

Existe otro caso posible en el bajo, donde los botones del lado izquierdo se tengan que pulsar de manera simultánea, ver Figura 5.12. En nuestro lenguaje se simbolizará con dos puntos en vez de las ya mencionadas comas o guiones para denotar esa simultaneidad.

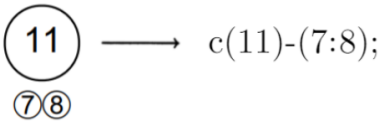


Figura 5.12: Representación del bajo simultáneo

Las decisiones tomadas para representar el bajo en el lenguaje intermedio son las siguientes:

- Se requerirá de un guión justo después de la sección de la melodía '-' para indicar que se quiere representar el bajo.
- Siempre se deberá abrir y cerrar el paréntesis con el contenido del bajo en el interior.
- Los botones del bajo que se quieran tocar por cada sonido de la melodía estarán separados por comas.
- Los botones del bajo que se quieran tocar consecutivamente junto a un sonido de la melodía estarán separados por guiones.
- Los botones del bajo que se quieran tocar de manera simultánea estarán separados por dos puntos.
- El ';' vendrá a continuación del paréntesis de cierre del bajo, no de la melodía.

Silencios

Los silencio musicales, también tienen una representación gráfica dentro de la escritura propia de la trikitixa. Estos se pueden ver tanto representados en la melodía como en el bajo. Es posible ver un ejemplo en la Figura 5.13, pudiendo distinguir de izquierda a derecha los siguientes casos:

- Un silencio único en la melodía.
- Un silencio en la melodía y el bajo

- Un silencio en la melodía y bajo alternado con silencios
- Un único silencio en el bajo
- Bajo alternado con silencios junto a melodía

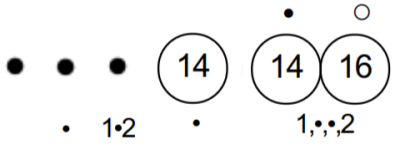


Figura 5.13: Representación de silencios en escritura de trikitixa

Aunque sean casos distintos, la implementación será la misma. Al silencio se le ha dado el valor '0'. Y con él, es posible realizar cualquiera de las combinaciones previamente mostradas (Figura 5.14).

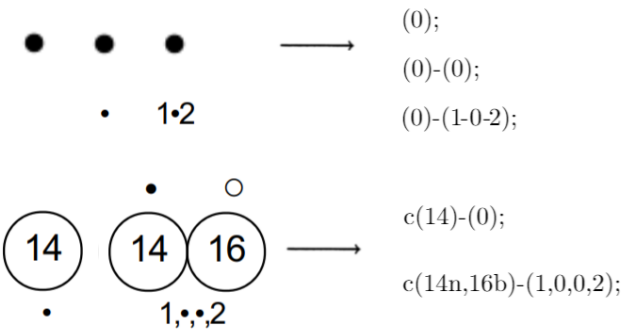


Figura 5.14: Representación de silencios en lenguaje intermedio

Es visible por tanto, que el silencio se integra como un número más dentro del lenguaje intermedio. No siendo necesario indicar el estado del fuelle si se quiere representar uno de estos en la melodía. Valiendo lo mismo denotar un único silencio en el bajo, cómo no hacerlo

Tresillos

Puede ocurrir, que el trikitilari quiera representar en la melodía un tresillo. Es decir, aunar en un tiempo 3 sonidos melódicos simultáneos. En la escritura de «zenbakiz» se suele representar como en la Figura 5.15:

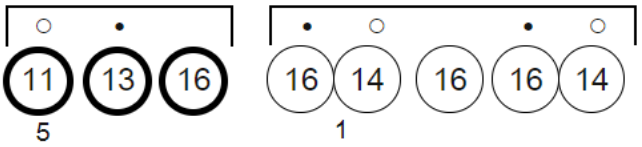


Figura 5.15: Representación de tresillos en la escritura de trikitixa

Hay que tener en cuenta entonces, que únicamente se van a representar 3 sonidos melódicos en el tresillo, y siempre suelen acompañarse de un único sonido del bajo. En la Figura 5.16 se muestra como se representan los tresillos en el lenguaje intermedio.

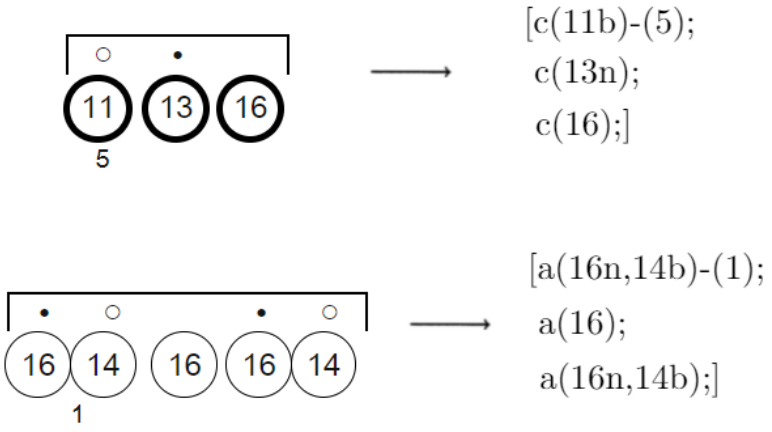


Figura 5.16: Representación de tresillos en el lenguaje intermedio

Viendo los ejemplos, es posible aclarar que se usarán corchetes de apertura y cierre para incluir en ellos los sonidos correspondientes al tresillo. Se tiene que tener en cuenta, que la escritura de los elementos internos será la misma que en ejemplos previos, y que solamente se deben incluir 3 conjuntos de sonidos melódicos.

Signos de repetición

En las propias partituras de acordeón existen diversos signos de repetición para indicar el orden de las repeticiones en cada una de las vueltas. De ellos se ha decidido adaptar el que hace función de 'Coda' (Figura 5.17).

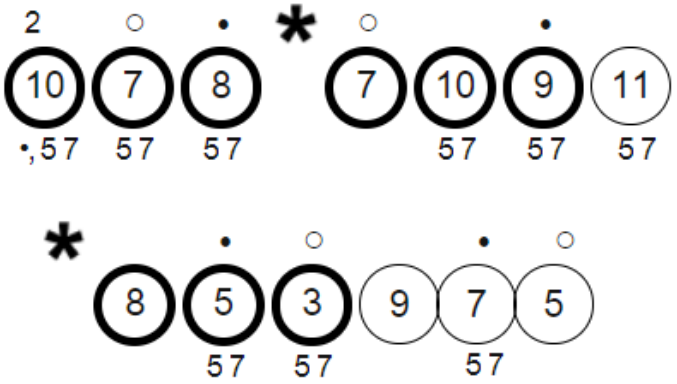


Figura 5.17: Ejemplo de 'Coda' en la escritura de trikitixa. El asterisco '*' indica el lugar de salto

En la Figura 5.18, se ve el uso del asterisco para identificar de donde a donde se debe saltar a la hora de realizar la repetición. En la primera vez se tocará todo lo incluido entre los dos asteriscos, pero una vez se repita se saltará de un asterisco a otro sin interpretar lo que hay entre ellos.

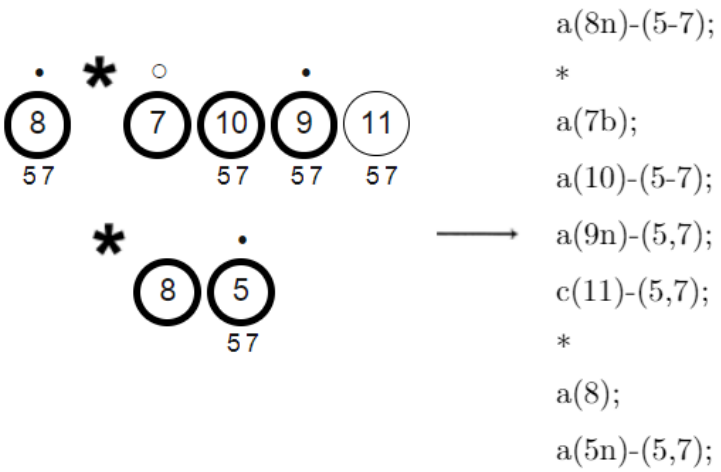


Figura 5.18: Ejemplo de 'Coda' en el lenguaje intermedio

Extrayendo una sección mostrada en la Figura 5.18, se ve como utilizando el símbolo '*' es posible tratar la 'Coda' de la misma manera que se utiliza en la partitura de trikitixa. Cabe decir que el símbolo de Coda únicamente tiene sentido apareciendo dos veces por partitura, esta regla se cumple también en el lenguaje intermedio.

Como complemento a la ‘Coda’ y aunque no esté explícito en la escritura de la trikitixa, se ha añadido el ‘Dal Segno’ o ‘Segno’. Este símbolo indica a donde debe saltarse una vez se llega al símbolo de la ‘Coda’ sin tener en cuenta repeticiones. Su símbolo musical es 𝄋 y en el lenguaje intermedio se ha implementado utilizando el signo ‘&’. Una representación en el lenguaje intermedio sería el de la Figura 5.19.

```
a(8n)-(5-7);
&
a(7b);
a(10)-(5-7);
*
c(11)-(5,7);
```

Figura 5.19: Ejemplo de ‘Segno’ en el lenguaje intermedio

Existen otro tipo de símbolos de repetición, basados en la repetición de partes más concretas. En la escritura de trikitixa, se representan como paréntesis, donde el paréntesis de apertura indica el inicio de la parte que se va a tener que repetir y el de cierre el final. Una vez llegado a ese final, se vuelve a la localización del paréntesis de apertura, siguiendo adelante sin volver a repetir. También es posible que aparezcan varias de estas repeticiones de manera consecutiva, como se ve en la Figura 5.20.

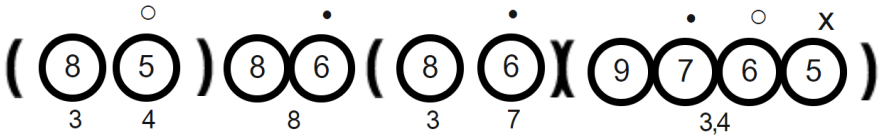
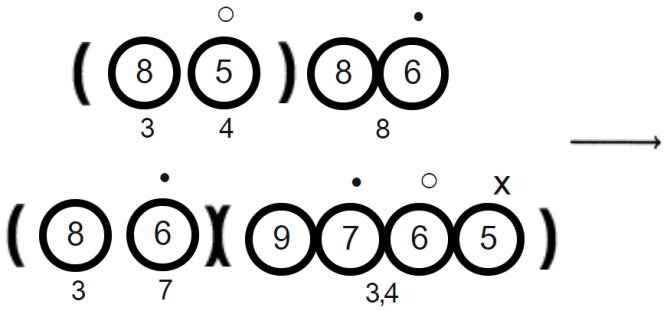


Figura 5.20: Ejemplo de signos de repetición de lenguaje numérico

En el lenguaje intermedio, estos símbolos van de manera independiente, al igual que la ‘Coda’. Se crean combinando símbolos de raya vertical ‘|’ y dos puntos ‘:’ (Figura 5.21). Cada uno significa repeticiones hacia un lado distinto, o ambos lados.




```
|:
a(8)-(3);
a(5b)-(4);
:|
a(8,6n)-(8);
|:
a(8)-(3);
a(6n)-(7);
:|:
a(9,7n,6b,5x)-(3,4);
:|
```

Figura 5.21: Signos de repetición en el lenguaje intermedio

También se ha añadido un símbolo adicional que sirve para la separación de secciones dentro de la partitura, el doble barra '||'. Su representación dentro de la partitura y la del resto de símbolos de repetición sería el de la Figura 5.22.

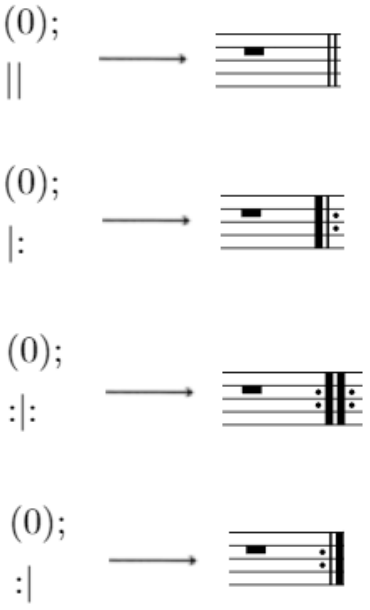


Figura 5.22: Signos de repetición en la partitura

5.2. Lilypond

Lilypond es una herramienta de edición de partituras basada en texto con gran cantidad de opciones. Lilypond se ha usado para crear una notación con la suficiente capacidad de representar el resultado equivalente de transformar una partitura de lenguaje de trikitixa a partitura tradicional. Esta notación se obtendrá una vez se compile el lenguaje intermedio, obteniendo cómo resultado la notación en Lilypond. En el siguiente apartado se quiere mostrar cómo será ese resultado obtenido, comparándolo con el propio lenguaje intermedio. Para más detalles sobre este tipo de notación consultar el Apéndice B.

Notación

A la hora de representar los dos pentagramas dentro de Lilypond, se han definido dos registros, correspondiendo cada uno a una de las voces del acordeón. El primer registro se ha denominado `pentasol`, ya que guarda la melodía escrita en Clave de Sol. El segundo registro denominado `pentafa`, dado que guarda el bajo en Clave de Fa.

Melodía

Para cada línea de la melodía del lenguaje intermedio, Lilypond guardará en el registro `pentasol` su correspondiente resultado. Es decir, que para cada línea terminada en punto y coma (salvo excepciones más adelante especificadas) se registrará esas notas de la melodía en un compás.

El compilador tendrá en cuenta en todo momento si para la melodía el fuelle se está abriendo o cerrando. En la partitura esto se verá reflejado a la notación que acompañará a la melodía, para así tener una referencia constante de la escritura de acordeón junto a la del pentagrama. A su vez se mostrará también la digitación correspondiente a esa nota. Se muestra la Figura 5.23 como entrada en lenguaje intermedio y salida en Lilypond.

a(5b);


```
penta_sol = { \penta_sol < a''\finger\markup{\override #'(thickness . 5)
\circle{5} \draw-circle #0.25 #0 ##f} > 1 }
```

Figura 5.23: Resultado de lenguaje intermedio a Lilypond con melodía

Es posible diseccionar el ejemplo en las siguientes partes:

- `penta_sol`: Variable donde se guardará el contenido de la nueva melodía. Entre llaves se incluirán todas las nuevas notas.
- `\penta_sol`: Llamada a la variable `penta_sol` ya existente para guardar las nuevas notas después de las anteriores, estas se guardarán entre signos de mayor y menor.
- `a''`: Nota de la melodía, concordando con la nota correspondiente al lenguaje intermedio. En este caso al ser el número 5 con el fuelle abriendo se sabe que la nota es 'La 5'.
- `\finger\markup{\override #'(thickness . 5) \circle{5} \draw-circle #0.25 #0 ##f}`: Se usa esta parte de la notación para poder representar la escritura de acordeón dentro de la partitura de pentagrama. Se determina el estado del fuelle cambiando el atributo 'thickness', siendo 5 para la representación del fuelle abriendo y 1 para cerrar. También se representa el número del lenguaje intermedio indicándolo en el atributo 'circle' y la digitación, en este caso el dedo anular.
- `1`: Indica la duración de la nota de la melodía, en este caso es una redonda, al no tener un ritmo definido e intentar representar únicamente el sonido de las mismas.

Es posible representar además, varias notas simultáneas en la melodía como bien se ha podido demostrar en el apartado del lenguaje intermedio como es posible ver en la Figura 5.24).

c(8,6n,4b);

```
penta_sol = { \penta_sol < a''\finger\markup{\override #'(thickness . 1) \circle{8}
" "} c''\finger\markup{\override #'(thickness . 1) \circle{6} \draw-circle #0.25 #0 ##t}
f'''\finger\markup{\override #'(thickness . 1) \circle{4} \draw-circle #0.25 #0 ##f} > 1 }
```

Figura 5.24: Resultado de lenguaje intermedio a Lilypond con melodía variada

Como se puede apreciar, la estructura de la sentencia es similar a la anterior, solo que para cada nueva entrada de una nota se añaden sus datos correspondientes. La duración de las notas se indica una única vez, cada vez que el compilador detecta una nueva línea de entrada del lenguaje intermedio.

- `a'' c'' f'''`: Estas letras corresponden a cada una de las notas que se representan tanto en el lenguaje intermedio como en Lilypond. Cada una de ellas da fin a la notación de la nota anterior para comenzar la suya propia.

- `\circle{8}` `\circle{6}` `\circle{4}`: Representan los círculos correspondientes al lenguaje de trikitixa que se mostrarán en la partitura, coincidiendo cada una de ellas con los número del lenguaje intermedio.

Bajo

Al igual que en la melodía, Lilypond guardará en un registro llamado `pentafa` el correspondiente resultado del bajo. Diferenciando así el apartado de la melodía del bajo dentro del fichero resultante '`.ly`'.

Quitando la parte de la Clave de Sol perteneciente a la melodía se tiene la capacidad de ver como resulta el bajo en Lilypond compilando el lenguaje intermedio en la Figura 5.25.

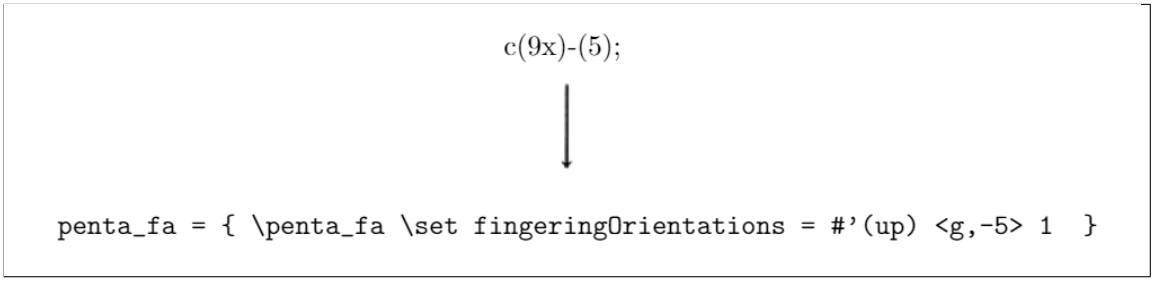


Figura 5.25: Resultado de lenguaje intermedio a Lilypond con bajo

Es posible dividir entonces el resultado en los siguientes apartados:

- `penta_fa`: Variable donde se guardará el contenido de el nuevo bajo. Entre llaves se incluirán todas las nuevas notas.
- `\penta_fa`: Llamada a la variable `penta_fa` ya existente para guardar las nuevas notas después de las anteriores.
- `\set fingeringOrientations = #'(up)`: Indicador de la orientación de la digitación, o en este caso los números correspondientes al bajo en la propia partitura.
- `<g,-5>`: Como en la melodía, se guardará entre signos de menor y mayor las notas correspondientes al bajo. Donde en este caso 'g,' es 'Sol 2' y el número 5 junto al guión representa el número que estará junto a la nota representando la escritura de acordeón.
- `1`: Indica la duración de la nota de la melodía, en este caso es una redonda, al no tener un ritmo definido e intentar representar únicamente el sonido de las mismas.

Asimismo se deberá representar el resto de opciones que presenta el bajo en el lenguaje original, por ejemplo, el guión y los dos puntos (Figura 5.26).

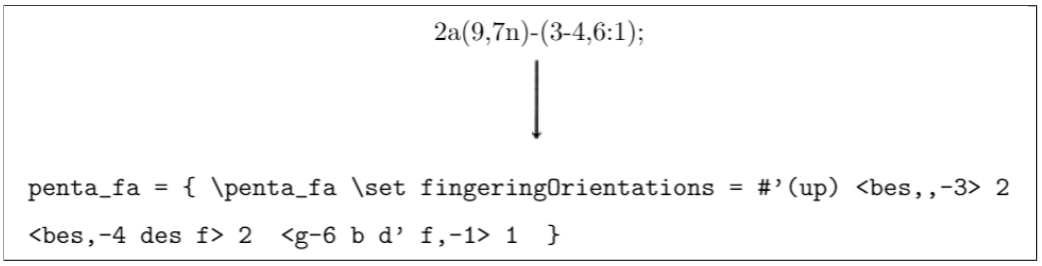


Figura 5.26: Resultado de lenguaje intermedio a Lilypond con bajo variado

Para diferenciar este extracto del anterior habría que fijarse en los siguientes apartados:

- `<bes,,-3>2 <bes,-4 des f>2`: La parte de Lilypond correspondiente a las notas del bajo con guión. Estas notas aparecerán en el mismo compás pero no simultáneamente, por ello por cada nota añadida se dividirá 1 entre ese número total de notas, consiguiendo así las medidas para que todo encaje en un compás.
- `<g-6 b d' f,-1>1`: Corresponde a las notas tocadas simultáneamente, por ello se aúnan dentro de signos de mayor y menor.

Silencios

Como es posible visualizar en Figura 5.27, para añadir silencios únicamente es necesario indicarlos con la letra 'r'.

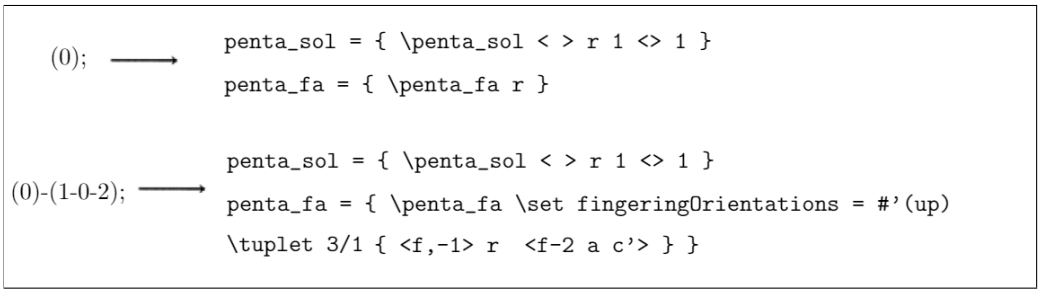
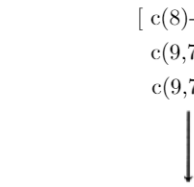


Figura 5.27: Resultado de lenguaje intermedio a Lilypond con silencios

Tresillos

Una vez el compilador ha leído qué notas deben estar dentro del tresillo indicado por los corchetes en el lenguaje intermedio, se usará la notación '\tuplet' junto a la equivalencia numérica (este caso siempre 1/3) para denotar que es un tresillo (Figura 5.28).

[c(8)-(1);
c(9,7b);
c(9,7b);]



```
penta_sol = { \penta_sol \tuplet 3/1 {
  < a''\finger\markup{\override #'(thickness . 1) \circle{8} " "}  >
  < e''\finger\markup{\override #'(thickness . 1) \circle{9} " "}
  g''\finger\markup{\override #'(thickness . 1) \circle{7} \draw-circle #0.25 #0 ##f}  >
  < e''\finger\markup{\override #'(thickness . 1) \circle{9} " "}
  g''\finger\markup{\override #'(thickness . 1) \circle{7} \draw-circle #0.25 #0 ##f}  >  } }
penta_fa = { \penta_fa \set fingeringOrientations = #'(up) <f,-1> 1  }
```

Figura 5.28: Resultado de lenguaje intermedio a Lilypond con tresillo

Signos de repetición

Cada signo de repetición tiene sus caracteres correspondientes. Para no causar errores de compilación cada uno de ellos se introduce como una entrada nueva tanto en la melodía con `pentasol` en la melodía y `pentafa` en el bajo. En la Figura 5.29 es posible ver todos estos ejemplos.

|:

```
penta_sol = { \penta_sol \bar ".|:" }
penta_fa = { \penta_fa \bar ".|:" }
```

:|:

```
penta_sol = { \penta_sol \bar ":..:" }
penta_fa = { \penta_fa \bar ":..:" }
```

:|

```
penta_sol = { \penta_sol \bar ":|." }
penta_fa = { \penta_fa \bar ":|." }
```

||

```
penta_sol = { \penta_sol \bar "||" }
penta_fa = { \penta_fa \bar "||" }
```

*

```
penta_sol = { \penta_sol \break }
penta_fa = { \penta_fa \break }
```

&

```
penta_sol = { \penta_sol \once \override Score.RehearsalMark.font-size = #3
\mark \markup { \musicglyph #"scripts.segno" } }
penta_fa = { \penta_fa \once \override Score.RehearsalMark.font-size = #3
\mark \markup { \musicglyph #"scripts.segno" } }
```

Figura 5.29: Resultado de lenguaje intermedio a Lilypond con signos de repetición

5.3. Flex & Bison

Para poder tratar el lenguaje intermedio definido en la Sección 5.1 como entrada y obtener su equivalente en Lilypond como se especifica en la Sección 5.2, se han utilizado Flex y Bison como base para crear un compilador capaz de realizar esta tarea. Flex se encargará del análisis léxico y Bison del sintáctico.

Procesador de lenguaje

Un compilador o procesador de lenguaje es a fin y al cabo, un programa (lenguaje máquina) capaz de leer de entrada un lenguaje (lenguaje fuente) y traducirlo a un programa o lenguaje equivalente (lenguaje destino). Es posible verlo simplificándolo a su mínima expresión en la Figura 5.30.

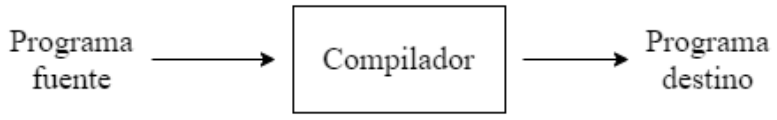


Figura 5.30: Ejemplo compilador

Si el programa de destino es un programa ejecutable en lenguaje máquina, es posible ejecutarlo para procesar entradas y crear salidas, es decir resultados. Aunque en el caso de este proyecto no se produce lenguaje máquina, si se puede decir que el proceso sería similar, ya que se pasaría de una entrada de fichero de texto (.txt) a una salida con un fichero en Lilypond (.ly) transformándolo en más adelante en PDF. En la Figura 5.31 se puede ver una representación.

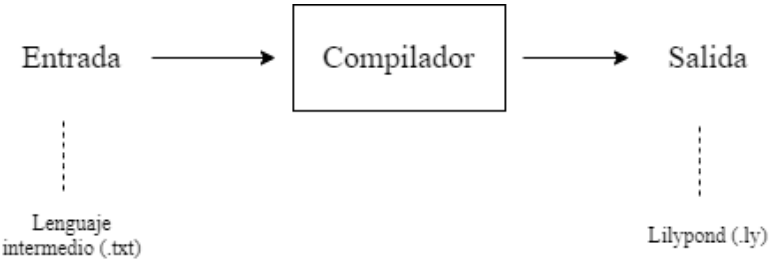


Figura 5.31: Ejemplo compilador en el proyecto

Aunque un programa más extenso pueda dividir su compilación en más apartados, como el *preprocesador*, *ensamblador*, *enlazador* etcétera [7], en este caso se simplificará con únicamente la parte del compilador y su propia estructura, detallándose en la Sección 5.4.

Analizador sintáctico (Parser)

Como procesadores de lenguaje que son Flex y Bison, van a ser útiles a la hora de querer leer un lenguaje para así obtener una equivalencia en un lenguaje requerido. Una de las mayores razones de haber elegido esta opción es que en concreto Bison genera un 'parser' (analizador sintáctico) LALR/LR (*Left-to-right, Rightmost derivation in reverse*). Significando esto la posibilidad de generar una gramática recursiva hacia la izquierda, cosa que no todos los tipos de traductores implementan. Como es por ejemplo el caso de ANTLR (*ANother Tool for Language Recognition*), generando 'parsers' LL (*Left-to-right, Leftmost derivation*), más complicados de escribir y testear.

Dado que la traducción dirigida por la sintaxis representa el núcleo de prácticamente cualquier *front-end* de un compilador moderno, el parser como implementación del análisis sintáctico, constituye una parte importante de cualquier compilador. Por lo tanto, el analizador sintáctico debe proporcionar al programador del compilador estas características:

(a) Un marco sólido para la evaluación de las acciones semánticas durante o después del análisis sintáctico.
(b) Un soporte adecuado para producir informes detallados de errores sintácticos.

Las dos técnicas predominantes para el análisis sintáctico de los lenguajes de programación son el análisis descendente y el análisis ascendente. Los analizadores sintácticos LL representan el núcleo del primer grupo y los analizadores LR el núcleo del segundo.

Ambas técnicas tienen sus ventajas y desventajas. El análisis sintáctico descendente proporciona *la legibilidad de las implementaciones de descenso recursivo (RD) del análisis sintáctico LL, junto con la facilidad de incorporación de acciones semánticas*, mientras que el análisis sintáctico ascendente es lineal en el tamaño de la gramática. [8].

Así es como se definen ambos analizadores sintácticos en el libro *"Construcción de compiladores"* de *Kenneth C. Louden* [9]:

- *Un algoritmo de análisis sintáctico descendente analiza una cadena de tokens de entrada mediante la búsqueda de los pasos en una derivación por la izquierda. Un algoritmo así se denomina descendente debido a que el recorrido implicado del árbol de análisis gramatical es un recorrido de preorden y, de este modo, se presenta desde la raíz hacia las hojas.*

- *Con una terminología similar a la de los analizadores sintácticos LL(I), el algoritmo ascendente más general se denomina análisis sintáctico LR(1). La L indica que la*

entrada, se procesa de izquierda a derecha, "Left-to-right", la R indica que se produce una derivación por la derecha, "Rightmost derivation", mientras que el número 1 indica que se utiliza un símbolo de búsqueda hacia adelante.

En la Tabla 5.3 se muestran de manera clara ambos tipos de analizadores y sus diferencias [10].

LL Parser	**LR Parser**
Conocido como el parser que va de 'arriba a abajo'	Conocido como el parser que va de 'abajo a arriba'
La primera L de LL es para la derivación de izquierda (left) a derecha (es decir, la entrada se procesa en el orden en que se lee) y la segunda L para la derivación de izquierda	La primera L de LR es para la derivación de izquierda (left) a derecha (es decir, la entrada se procesa en el orden en que se lee) y R (right) es de derivación a la derecha (right)
LL comienza únicamente con la raíz no-terminal	LR termina únicamente con la raíz no-terminal
LL termina cuando la pila está vacía	LR comienza con una pila vacía
LL expande los conjuntos no-terminales	LR reduce el conjunto no-terminal
Durante el análisis sintáctico LL, el analizador sintáctico elige continuamente entre dos acciones. **Predecir**: Basado en el conjunto no-terminal más a la izquierda y en un cierto número de tokens de búsqueda. **Emparejar**: Hace coincidir el símbolo terminal más a la izquierda con el símbolo no-terminal más a la izquierda de la entrada.	Durante un análisis sintáctico LR, el analizador sintáctico elige continuamente entre dos acciones. **Desplazamiento**: Añade el siguiente token de entrada a un buffer para su consideración. **Reducir**: Reduce una colección de terminales y no terminales.
Los analizadores sintácticos LL son más fáciles de escribir pero son menos potentes. Vienen en varias versiones como LL(1), etc.	Los analizadores sintácticos LR son muy potentes. Vienen en muchas versiones como LR(0), SLR(1), LALR(1), LR(1), etc.

Tabla 5.3: Comparación entre analizadores sintácticos LL y LR

Se ha decidido optar por un analizador sintáctico 'LR'. La razón principal ha sido su capacidad de procesar texto recursivamente a la izquierda, sin tener la necesidad de declarar operadores, simplificando así en gran medida la elaboración. Esto hace que el analizador sintáctico 'LR' pueda manejar más tipos de gramática, ya que el 'LL' solo llega a mirar los primeros tokens por regla y 'LR' lleva a ver todos. Además hace posible la creación de lenguajes menos ambiguos.

Estructura del compilador

Los compiladores se forman por dos procesos principales, **análisis** y **síntesis**.

Análisis: La parte correspondiente al análisis divide el programa fuente, en nuestro caso el lenguaje intermedio, en componentes, obligando una estructura gramatical. A continuación usando la estructura creada la transforma en una representación intermedia del lenguaje. Dado que el usuario es quien introduce el texto a compilar, pueden detectarse inconsistencias semánticas, sintácticas y en general malformaciones en el léxico introducido. Proveyendo al usuario de la información correspondiente para solucionarlos. Durante todo este proceso, se recolecta información sobre el programa fuente, guardando una estructura de datos propia llamada tabla de símbolos. Esta pasa a la fase de síntesis junto a la representación intermedia previamente mencionada.

Síntesis: En el apartado de la síntesis se construye el programa de destino deseado. Valiéndose de la representación intermedia y de la información de la tabla de símbolos.

Es posible examinar con más detenimiento cada uno de estos procesos, es posible ver que están divididos en diferentes fases realizándose en cada una distintas operaciones lógicas. Cada una de estas etapas forman parte del proceso lineal de un compilador. Pueden venir tanto de manera independiente como agrupada y el resultado de todas estas no tiene por qué darse dentro del compilador de manera explícita. Todo esto se ve mejor reflejado en la Figura 5.32.

Como se puede distinguir en la Figura 5.32, algunos compiladores tienen una fase de optimización de código. Este se ubica entre el *front-end* y el *back-end*. El objetivo de esta fase es optimizar la transformación sobre la representación intermedia para que el *back-end* generé por su lado un programa destino mejor de lo que sería sin optimizar. Este apartado es opcional, ya que no todos los compiladores lo tienen. No se darán detalles para el compilador aquí desarrollado, pero sí se explicará esta fase más adelante.

Análisis léxico

La primera etapa del compilador, la considerada como escáner o analizador léxico, se encarga de leer el programa fuente, identificar caracteres y agruparlos en secuencias con significado conocidas como lexemas. En cada uno de estos lexemas el analizador produce un *token* y se lo asigna:

(*nombre-token*, *valor-atributo*)

pasando a la siguiente fase, el análisis sintáctico. En este caso el *nombre-token* hace referencia a un símbolo abstracto utilizado durante el análisis sintáctico, y el *valor-atributo* apunta a una entrada correspondiente en la tabla de símbolos de este token, necesario para el análisis semántico.

En este caso, se ha utilizado Flex como analizador léxico. Flex (la versión más rápida de Lex) es una herramienta para generar escáneres. A continuación se expondrá un ejemplo correspondiente al lenguaje intermedio creado previamente en el Subcapítulo (5.1).

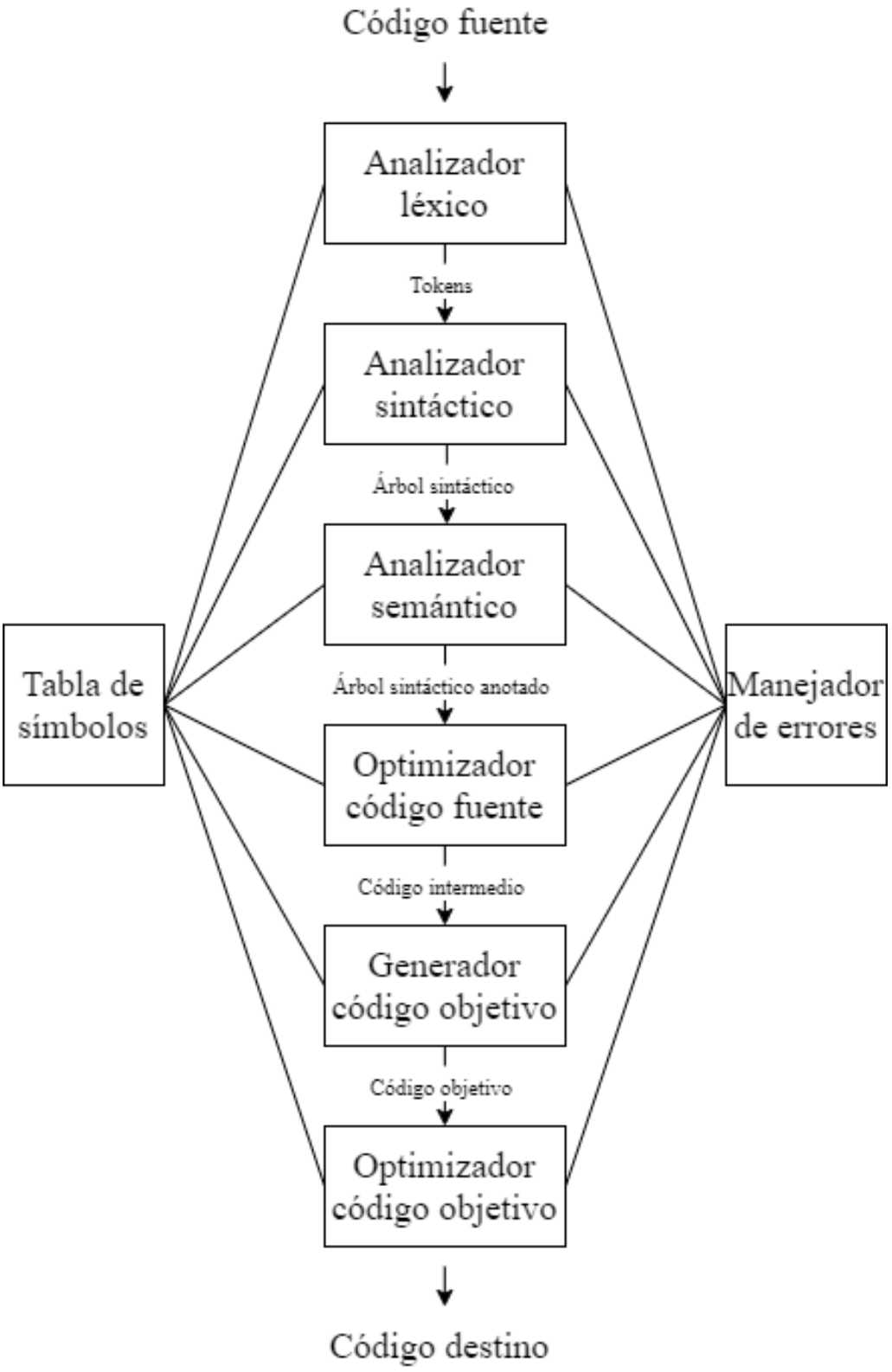


Figura 5.32: Fases del compilador

```
c(9n,7b)-(7-8,7-8);
```

Es posible agrupar los caracteres aquí mostrados de la manera que lo haría el compilador en un análisis léxico, teniendo en cuenta el léxico reflejado en el Apéndice C:

1. A `'c'` en el caso de este compilador, se le asigna el token `FUELLE_CERRAR`. Ya que `'c'` es un lexema que únicamente corresponderá a la indicación de que el fuelle está cerrado. El token será el identificador de la tabla de símbolos y `'c'` su valor. Al estar contemplado por el analizador léxico no se considera símbolo abstracto.
2. El símbolo de paréntesis de apertura `'('` es un lexema al cual se le asigna un token `PARENTESIS_ABRIR`. A todos los símbolos no abstractos es posible asignarles cualquier token, pero por conveniencia y por como se ha desarrollado el apartado Flex se ha decidido representarlo así.
3. Se le asigna el token `PARENTESIS_CERRAR` al símbolo de paréntesis de cierre `)` considerado lexema. A ambos paréntesis se les asignará el mismo token, teniendo valores distintos dependiendo de su posición.
4. A los números que aparecen en la representación, tanto los bajos como los agudos (`'9 7 8'`), se les asigna un token diferente aunque sean el mismo número repetido. La estructura básica de un token es (*nombre-token*, *valor-atributo*), transformándose en este caso para la melodía `<numbers, 1>` para el '9' y `<numbers, 2>` para el '7'. En el caso del bajo `<numbers, 3>` para el primer '7', `<numbers, 4>` para el primer '8', `<numbers, 5>` para el segundo '7' y `<numbers, 6>` para el segundo '8'.
5. Las comas `','` cubrirán distintas funciones dependiendo de su posición en la gramática del analizador sintáctico, pero se les asignará a ambas el mismo token `COMA`.
6. Los guiones `'-'` cubrirán también distintas funciones dependiendo de su posición en la grámatica del analizador sintáctico, pero se les asignará a ambos el mismo token `GUION`.
7. Tanto la letra `'n'` como `'b'` están relacionadas con la digitación del acordeón, siendo pues sus tokens correspondientes `DIGIT_NEGRO` y `DIGIT_BLANCO` respectivamente.

Así es como resultaría tras el proceso de análisis léxico la cadena de símbolos y caracteres previa:

```
(FUELLE_CERRAR) (PARENTESIS_ABRIR) (numbers, 1) (DIGIT_NEGRO) (COMA) (numbers, 2)
(DIGIT_BLANCO) (PARENTESIS_CERRAR) (GUION) (PARENTESIS_ABRIR) (numbers, 3)
(GUION) (numbers, 4) (COMA) (numbers, 5) (GUION) (numbers, 6) (PARENTESIS_CERRA)
(PUNTOCOMA)
```

Análisis sintáctico

La segunda fase del compilador, consiste en un análisis sintáctico o *parsing* utilizando los tokens creados por el analizador léxico, creando una representación intermedia, un árbol sintáctico a partir de la gramática creada por el analizador léxico. Dicha estructura determinará la estructura gramatical del flujo de tokens. Determinando los elementos estructurales del programa y sus relaciones, en el cual cada nodo interior representa una operación y los hijos del nodo representan los argumentos de la operación.

En el árbol se muestra el orden en el que deben llevarse a cabo las operaciones de la siguiente asignación:

```
c(9)-(7-8);
```

Se sabe por el anterior apartado que su resultado en tokens sería el siguiente:

```
(FUELLE_CERRAR) (PARENTESIS_ABRIR) (numbers, 1) (PARENTESIS_CERRAR) (GUION)
(PARENTESIS_ABRIR) (numbers, 2) (GUION) (numbers, 3) (PARENTESIS_CERRA)
(PUNTOCOMA)
```

Utilizando la gramática elaborada para el lenguaje intermedio, la cual es posible visualizar en el Apéndice C, se ha llevado a cabo la transformación de la sentencia obtenida tras el análisis léxico a un árbol sintáctico. Para ilustrar de una manera más clara el proceso de transformación, es posible reflejarlo en la Figura 5.33.

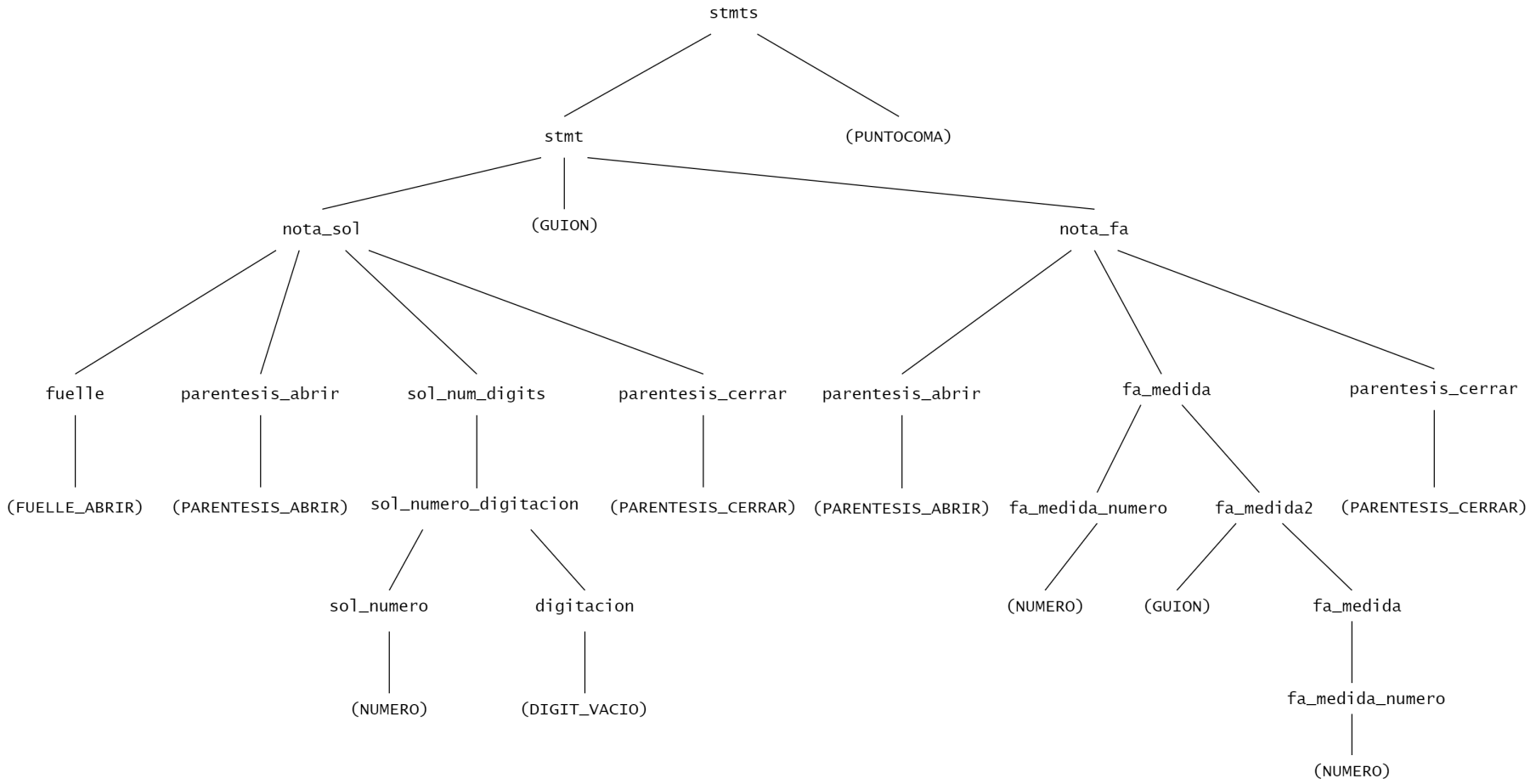


Figura 5.33: Árbol sintáctico

La transformación a árbol se podría resumir en la siguiente manera:

- El árbol tiene un nodo interior llamado `stmts`, que incluye todo el conjunto de caracteres y símbolos de todas las expresiones que se incluirán en el árbol. En este caso únicamente se introduce una expresión. Sus dos hijos, uno terminal llamado `(PUNTOCMA)` y otro `stmt` el cual es la representación del conjunto de caracteres y símbolos de una única expresión `c(9)-(7-8);`. Cada una de estas sentencias incluyen una melodía (hijo `nota_sol`) y puede que un bajo (hijos `(GUION)` y `nota_fa`). Analizando estos dos hijos no terminales es posible sonsacar el resto de la estructura del árbol.
- Como `nota_sol` hace referencia a la melodía, tendrá hijos representando el `fuelle`, `parentesis_abrir`, `sol_num_digits` (equivalente a las notas) y `parentesis_cerrar`. En este caso `fuelle` tiene un hijo terminal `(FUELLE_ABRIR)`, `parentesis_abrir` un hijo terminal `(PARENTESIS_ABRIR)` y `parentesis_cerrar` un hijo también terminal `(PARENTESIS_CERRAR)`. El único que no tiene hijo terminal es `sol_num_digits`, teniendo un hijo no-terminal `sol_numero_digitacion` con dos descendientes `sol_numero` y `digitación`. Siendo el hijo terminal del primero y representando el número de las notas correspondientes con `(NUMERO)`, y siendo el hijo terminal del segundo y representando la digitación para esas notas con `(DIGIT_VACIO)`.
- Por otro lado `nota_fa` referencia al bajo. Teniendo en este caso hijos representando el `parentesis_abrir`, `fa_medida` (equivalente a las notas y acordes) y `parentesis_cerrar`. En este caso el nodo `parentesis_abrir` tiene un hijo terminal `(PARENTESIS_ABRIR)` y `parentesis_cerrar` un hijo también terminal `(PARENTESIS_CERRAR)`. Siendo `fa_medida` el nodo no terminal con hijos `fa_medida_numero` y `fa_medida2`. El primero hace referencia a las notas y acordes con su hijo terminal `(NUMERO)`, y el segundo a los símbolos entre los números del bajo con su hijo terminal `(GUION)`. El resto de hojas son las llamadas anidadas a más números dentro del bajo.

Para proceder al análisis sintáctico en este compilador con la gramática previamente mencionada, se ha hecho uso de Yacc/Bison como generadores de este tipo de analizadores. Ambos son programas que reconocen la estructura gramatical de los programas. Bison es una versión más rápida de Yacc. Más adelante se darán detalles sobre el funcionamiento de Yacc/Bison y su interacción con Lex/Flex.

Análisis semántico

En un programa, la semántica es el "significado" de cada parte del lenguaje utilizado, en contra de la sintaxis, que se considera la "estructura". Estos significados se verán durante la ejecución del propio programa destino, pero es posible ir determinándolos según el compilador va tratando el lenguaje de entrada.

El analizador semántico se vale del árbol sintáctico previamente creado junto con la tabla de símbolos para comprobar la consistencia del programa fuente. Este proceso guarda la información de los tipos de símbolos y caracteres, tanto en la tabla de símbolos como en el árbol semántico para utilizarlo en la creación del código intermedio. Este código intermedio es el paso previo a la obtención del código objetivo del compilador.

Es posible dar un ejemplo con el mismo caso anterior:

```
c(9)-(7-8);
```

Comprobando el árbol sintáctico y la tabla de símbolos, el analizador semántico determina los valores y tipos de la expresión. Obteniendo un resultado como el de la Figura 5.34.

En este caso se identifican los números utilizados en la expresión como enteros, ya que se contemplan dentro del rango de números posibles para el uso coherente del lenguaje de este compilador. Tendrán un trato distinto al resto de tokens, que se guardarán como identificadores, no es necesario indicarlo en el árbol, pues será la misma información guardada de la tabla de símbolos con la estructura (*nombre-token*, *valor-atributo*). Por tanto, se anota cada token no-entero con su función dentro del léxico.

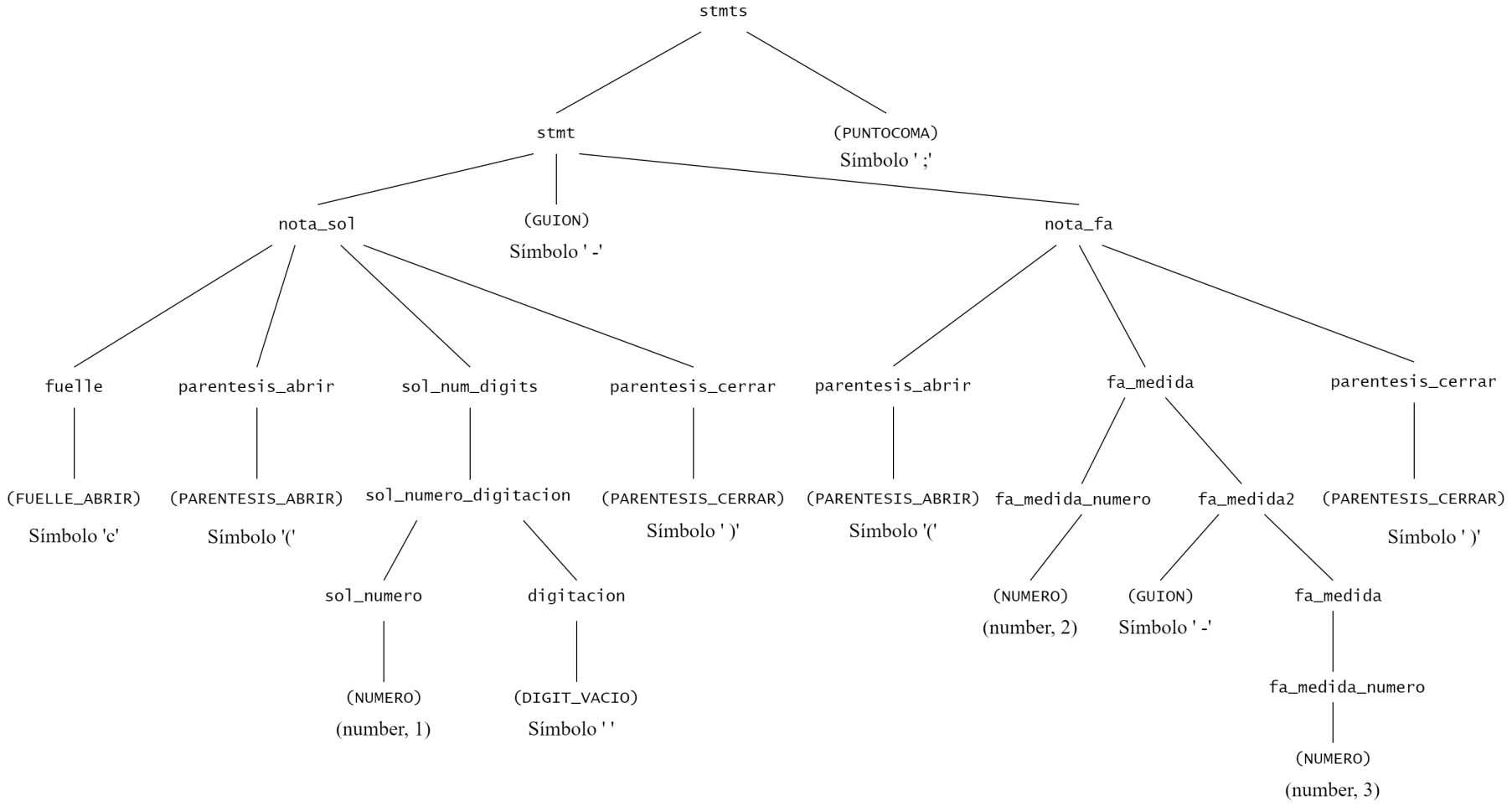


Figura 5.34: Árbol semántico

Optimización de código fuente

En la fase de optimización de código fuente, se trata de mejorar el código intermedio a obtener, de manera que produzca un código de destino mejor. Generalmente, eso suele significar más velocidad o un código más corto, también uno que consuma menos. El optimizador aplica transformaciones que preservan la semántica al árbol de análisis sintáctico anotado para simplificar su estructura y facilitar la generación de un código más eficiente.

En el caso de este compilador, encargado de pasar el lenguaje intermedio creado para los acordeonistas a un lenguaje Lilypond, la optimización no será muy grande, ya que el lenguaje no da opción a mucha abstracción ni opciones de definir una misma operación de distintas maneras. Únicamente es posible ver como puede que el optimizador retire los paréntesis de apertura y cierre, o los símbolos de punto y coma cada vez que se lea una línea del fichero de entrada:

```
c9-7-8
```

Generación de código objetivo

El generador de código transforma el árbol de análisis sintáctico anotado simplificado en código objeto utilizando reglas que denotan la semántica del lenguaje fuente. El generador de código puede integrarse con el analizador sintáctico. Transformándose en este caso en el código Lilypond objetivo. Siendo el del ejemplo antes utilizado:

```
penta_sol = {\penta_sol <e''\finger\markup{\override #'(thickness . 1)\circle9 " "} >1 }
penta_fa = {\penta_fa\set fingeringOrientations = #'(up) <c,-7>2 <c-8 e g>2 }
```

Optimización de código objetivo

En la última fase, el compilador intenta mejorar el código objetivo, generado previamente por el generador de código. Estas posibles mejoras incluyen la selección de modos de direccionamiento para la mejora de rendimiento, reemplazando así instrucciones lentas por otras más rápidas pero con la misma función, y eliminando redundancias en las operaciones.

En el caso de Bison, el optimizador "peep-hole" examina el código objeto, unas pocas instrucciones a la vez, e intenta realizar mejoras de código dependientes de la máquina. Más concretamente, en el caso que aquí ocupa, se encargará de eliminar caracteres redundantes o vacíos, así como llamadas innecesarias a las variables `penta_sol` y `penta_fa`.

Proceso final

Una vez ya obtenido el resultado en Lilypond, es posible representar la figura de las fases del compilador (Figura 5.32) en una manera más concreta para este proyecto (Figura 5.35).

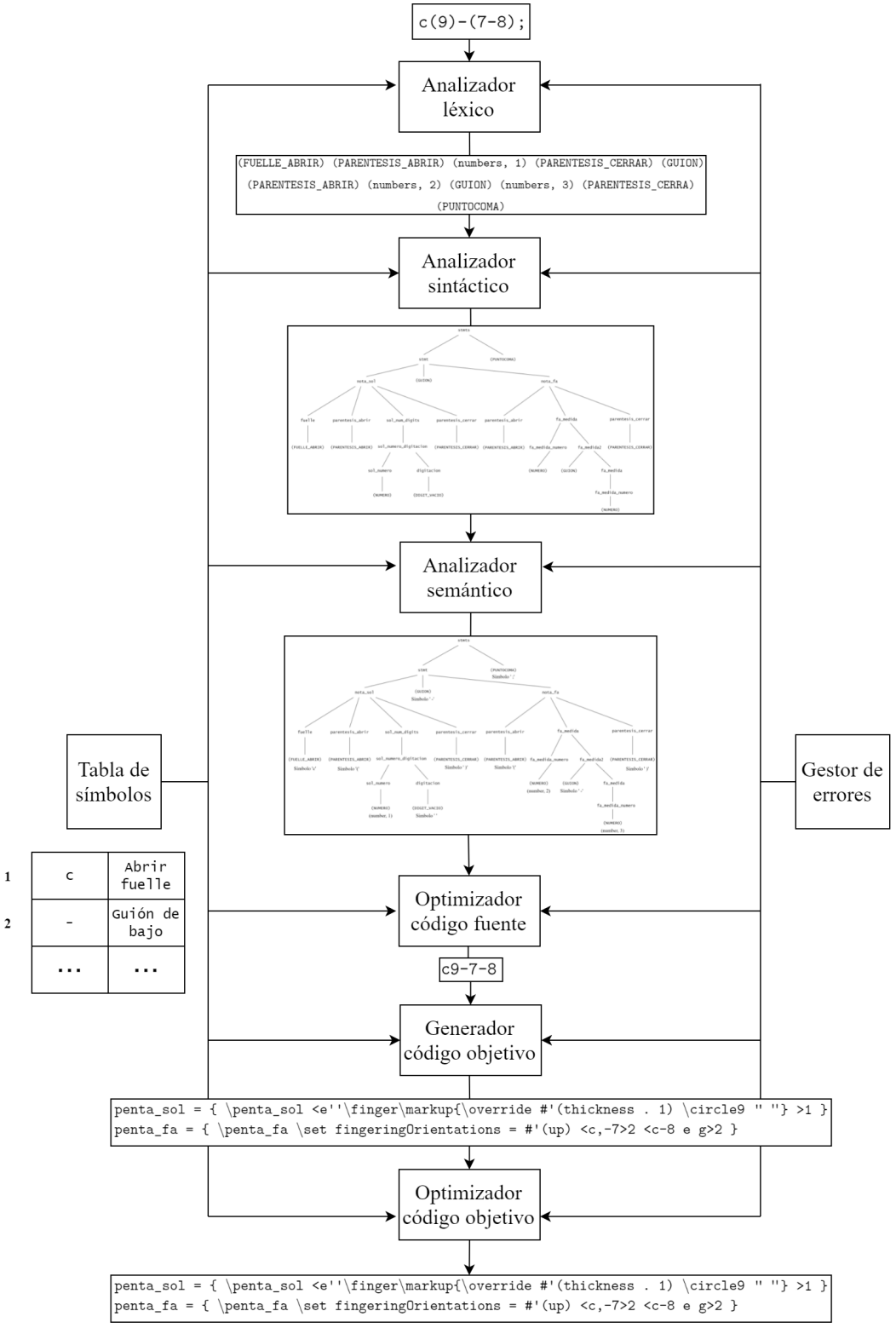


Figura 5.35: Fases del compilador detallado

5.4. Ejecución

Habiendo detallado la función y proceso de compilación, esta sección se enfocará en el proceso de ejecución propia de Flex y Bison. Además, se concretará la relación entre ambos analizadores y la generación de archivos que acarrean. Antes de proceder con las herramientas principales utilizadas en este proyecto, se darán unos detalles del funcionamiento de Lex y Yacc, ambas herramientas predecesoras de las ya mencionadas Flex y Bison.

Lex/Flex

Aunque el generador de analizadores léxicos aquí utilizado ha sido Flex, este es una alternativa a Lex, con la cual es compatible por completo y comparte prácticamente todas sus características, salvo contadas excepciones [11].

Su funcionamiento es el siguiente, Lex genera un código fuente en C partiendo de las especificaciones escritas en lenguaje Lex. Este código C generado contiene la función `yylex()`, la cual localiza lexemas que se ajusten a los patrones léxicos ya especificados en el código fuente Lex, realizando así acciones asociadas a cada uno de estos patrones. `yylex()` es capaz de llevar a cabo cualquier tipo de acción ante un patrón determinado, comportándose como un analizador léxico [12]. Para ilustrarlo se refleja en la Figura 5.36 (**NOTA**: algunos de los ficheros aquí mostrados se detallarán más adelante).

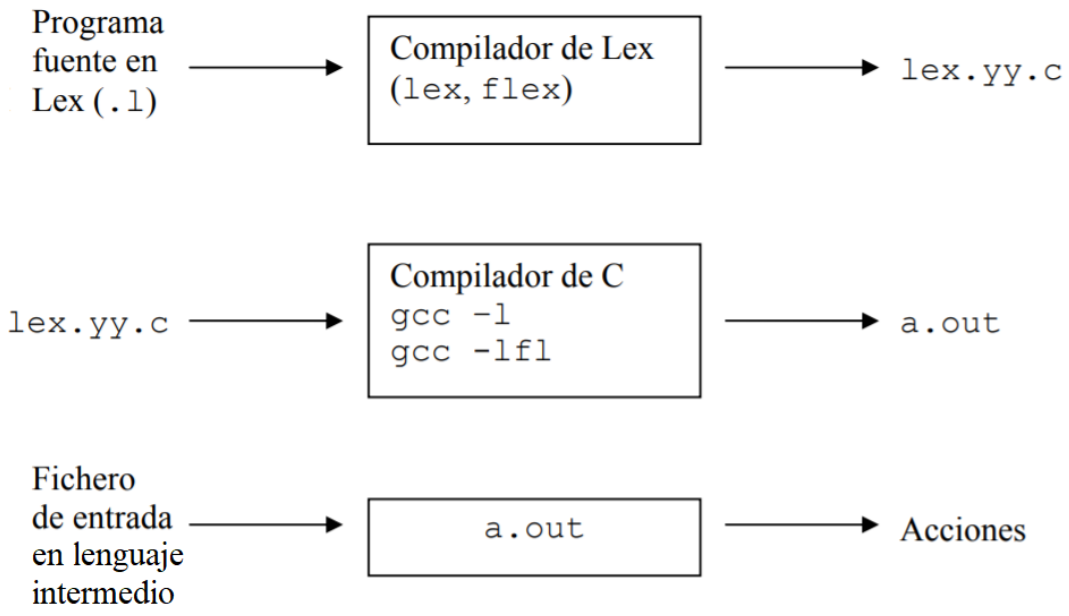


Figura 5.36: Funcionamiento Lex

Para que Lex/Flex reconozca patrones en el texto, el patrón debe estar descrito por una expresión regular. La entrada de Lex/Flex es un conjunto de expresiones regulares legibles por la máquina. La entrada se presenta en forma de pares de expresiones regulares y código C, denominados reglas. Lex/Flex genera como salida un archivo fuente en C, `lex.yy.c`, que define una rutina `yylex()`. Este archivo se compila y enlaza con la biblioteca `-lfl` para producir un ejecutable. Cuando es ejecutado, analiza su entrada en busca de ocurrencias de las expresiones regulares. Cada vez que encuentra una, ejecuta el código C correspondiente. Es así como se enlazan tokens con valores del programa fuente. El analizador léxico en este caso ignorará los espacios en blanco que separan los lexemas.

Cada vez que se llama a `yylex()`, este continúa procesando tokens desde donde lo dejó por última vez hasta que llega al final del archivo o ejecuta un retorno. Sin embargo, una vez que llega al final del archivo, cualquier llamada posterior a `yylex()` simplemente regresará inmediatamente [13].

Yacc/Bison

Igual que sucedía con Lex, Yacc no ha sido el analizador sintáctico que se ha utilizado para el trabajo. Pero su explicación es igualmente útil que la de Bison, ya que Yacc es completamente compatible con el Bison.

En cuanto a su funcionamiento es el siguiente. Yacc parte de un fichero fuente en el que se genera un fichero fuente en C. Este fichero contiene el analizador sintáctico. Aun así, un analizador creado de esta manera no es funcional por sí mismo, necesita de un analizador léxico externo. Así que en el fichero fuente generado en C previamente se contienen llamadas a la función `yylex()` que deberá estar definida devolviendo el tipo del lexema encontrado. Además de esto, se incorpora una función `yyerror()`, para cuando el analizador sintáctico encuentre un símbolo que no encaje con la gramática definida [14]. Para ilustrarlo se refleja en la Figura 5.37 (**NOTA**: algunos de los ficheros aquí mostrados se detallarán más adelante).

Para que el análisis comience la función `yyparse` es llamada. Esta, lee tokens, ejecuta acciones necesarias y finalmente se retorna cuando llega al final del fichero o al encontrar un error sintáctico del que no pueda recuperarse. `yyparse` devolverá 0 si el análisis tuvo éxito y 1 si se encontró un fallo [15]. Esto es útil a la hora de realizar pruebas sin tener un tratamiento de errores muy extenso, o si el compilador creado no es muy complejo.

Proceso de ejecución

Para generar el compilador de manera correcta, es necesario ejecutar los comandos correspondientes en consola con sus invocaciones pertinentes. En este aspecto la decisión ha sido generar

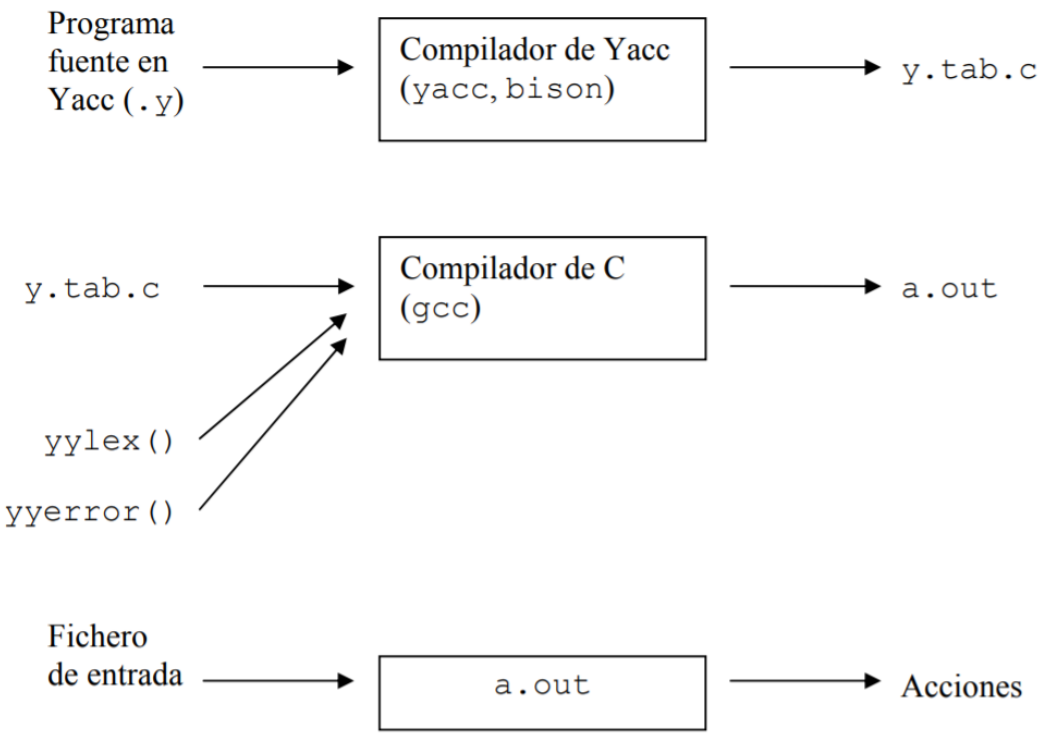


Figura 5.37: Funcionamiento Yacc

un `Makefile` donde se llevará a cabo toda la compilación. Esta compilación tendrá los siguientes comandos:

```
zenb2pent: compilador.l compilador.y
 bison -t -d compilador.y
 flex -d compilador.l
 gcc -o $@ compilador.tab.c lex.yy.c -lfl
```

Ejecutando en la línea de comandos `make` sucederá lo siguiente:

- Se creará el archivo `zenb2pent`, el cual será el resultado final de la compilación. Este archivo es al fin y al cabo el compilador que se utilizará para pasar un fichero en el lenguaje intermedio creado en este proyecto a Lilypond. El archivo `compilador.l` corresponde a Flex y `compilador.y` corresponde a Bison.
- Con Flex se invocará a `-d`, haciendo que se ejecute en modo de depuración. Esto será útil a la hora de hacer pruebas y poder tratar casos como el de los "espacio en blanco"del fichero de entrada. La salida de Flex es el fichero `lex.yy.c`, que contiene la función de análisis `yylex()`, varias tablas usadas por esta para emparejar tokens y unas cuantas rutinas auxiliares y macros.

- A la hora de ejecutar Bison se invocará a `-t`, produciendo una definición de la macro `YYDEBUG` en el archivo del analizador, para compilar las facilidades de depuración. Y también `-d`, que generará el archivo con las definiciones de tokens que necesita Flex. Se creará un fichero en C llamado `nombre_fuente.tab.c`. Por compatibilidad con Yacc, existe la opción `-y` que fuerza a que el archivo de salida se llame `y.tab.c`. Este archivo se incluye generalmente en la sección de declaraciones del fuente de Flex.
- Para finalizar con la compilación se compilan tanto el analizador léxico `lex.yy.c` como el sintáctico `y.tab.c` con el comando `gcc`. Estos pasos generarán un ejecutable llamado `zenb2pent`, permitiendo comprobar qué palabras pertenecen al lenguaje generado por la gramática descrita en el fichero Bison [16].

Es posible ilustrar todo este proceso con la Figura 5.38.

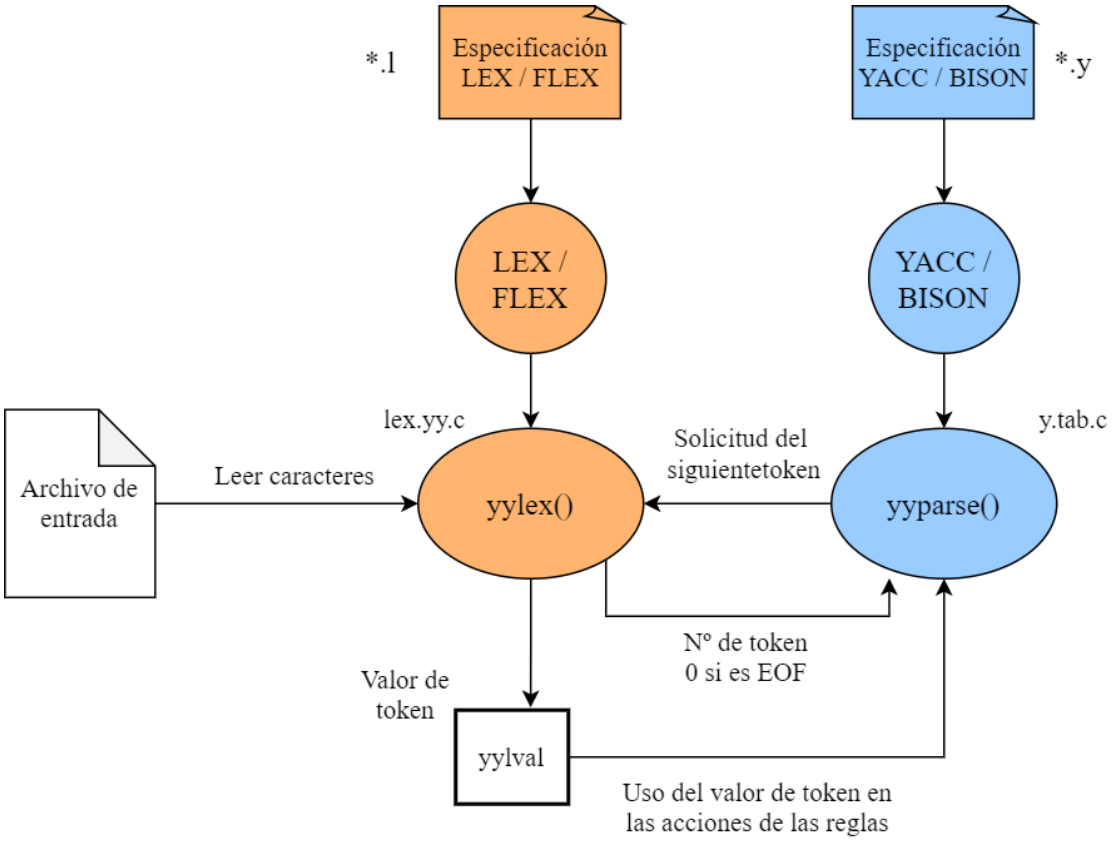


Figura 5.38: Interacción entre Flex y Bison

Habiendo obtenido ya el compilador, únicamente es necesario usarlo para tratar un archivo escrito en el lenguaje intermedio, para así poder transformarlo en Lilypond y a continuación en

PDF. Para ello valdrá un script como este:

```
#!/bin/bash
./zenb2pent < tfg.znb > partitura.ly
lilypond partitura.ly
```

Utilizando el compilador `zenb2pent`, el fichero de entrada en lenguaje intermedio `partitura.znb` se obtiene el fichero en anotación Lilypond `partitura.ly`, pudiendo transformarlo en el PDF con los requisitos solicitados (Figura 5.39). A la hora de hacer pruebas, se valdrá de esta transformación para la visualización de errores musicales que el compilador no contempla.

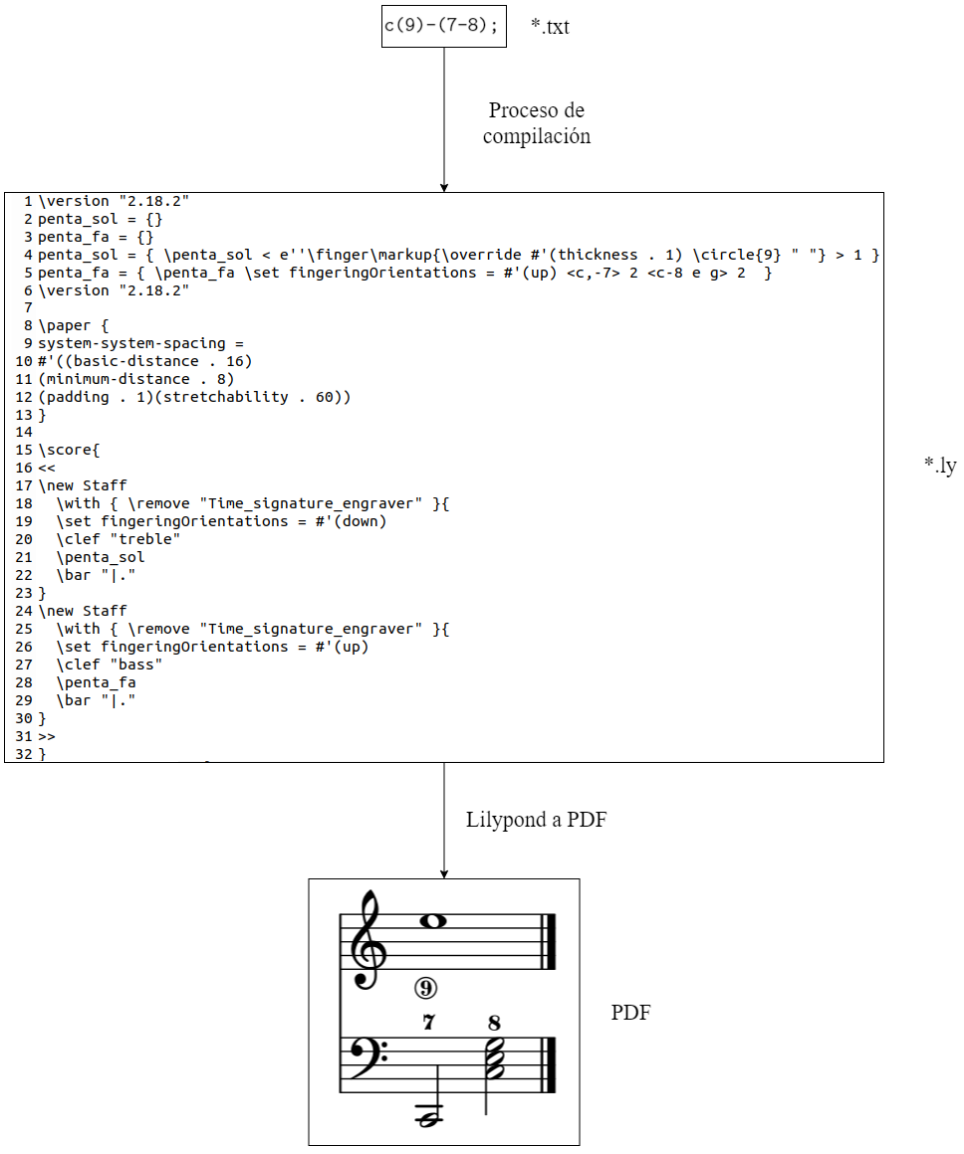


Figura 5.39: Resumen del proceso de compilación

CAPÍTULO 6

WEB: DJANGO

El siguiente capítulo servirá como explicación para la implementación y desarrollo de la web puesta en funcionamiento en el framework Django. También se expondrán sus funcionalidades principales y cómo llegar a ellas.

6.1. Instalación

Para poder desarrollar la web de manera correcta, por compatibilidad y sencillez de instalación, se ha decidido desarrollar la implementación con Django en Ubuntu. Una de las mayores razones de esta elección es debido a que la consola de GNU/Linux simplifica en gran medida las operaciones a realizar con el compilador.

Como primer paso es necesario instalar Python, tanto para la ejecución de ficheros '.py' como para la instalación de paquetes compatibles. Eso se ha realizado con el comando:

```
sudo apt-get install python3
```

Para llevar a cabo un correcto control de versiones, se ha utilizado e instalado Git. Esto se lleva a cabo con el comando:

```
sudo apt-get install git
```

Para poder mantener las dependencias requeridas para este proyecto de manera separada, se ha creado un entorno virtual de Python, donde se han instalado todas las librerías necesarias, así como el propio Django. Para instalarlo y ponerlo en marcha se utilizan los comandos:

```
sudo apt-get install python3-venv
python3 -m venv myenv
source myenv/bin/activate
```

Finalmente, para la instalación de Django se ha requerido de un fichero `requirements.txt` donde indicaremos la necesidad de instalar Django y su versión. Este fichero se encuentra en el mismo directorio donde se realiza el siguiente comando:

```
pip install --upgrade pip
```

Mediante este proceso ya se podría empezar a desarrollar el proyecto en Django y editar el código con el IDE (Integrated Development Environment) que más corresponda.

6.2. Ficheros

Django sigue una estructura de ficheros para organizar de manera correcta las aplicaciones de la web. En este caso se ha tenido en cuenta también el entorno virtual para reflejar de manera completa este apartado del proyecto. Es posible ver la estructura de fichero en la Figura 6.1. Mencionar que se ha hecho uso de la extensa documentación provista en la web oficial de Django para la elaboración de esta aplicación [17]. A continuación se expondrán los apartados más relevantes.

`manage.py`

Este archivo se utiliza básicamente como una utilidad de línea de comandos y para desplegar, depurar o ejecutar la aplicación web. Contiene código para ejecutar `runserver`, `makemigrations` o `migraciones` que utilizamos en el `Shell`. De entre todos ellos utilizaremos principalmente el primero para ejecutar el servidor de la aplicación web. Cabe mencionar que este fichero no deberá ser modificado.

`__init__.py`

Este archivo permanece vacío y está presente sólo para indicar que este directorio en particular (en este caso compilador) es un paquete. A este fichero tampoco se le deben aplicar cambios.

`setting.py`

Este archivo está presente para añadir todas las aplicaciones necesarias. Incluye la información sobre plantillas, ficheros estáticos y directorios necesarios dentro del proyecto. Es en términos generales, el archivo principal de la aplicación.

urls.py

Archivo utilizado para la gestión de URLs dentro de la aplicación. Como en este proyecto únicamente se dispone de una plantilla, es decir una URL, solamente se indica una dentro de la variable `urlpatterns`.

wsgi.py

Se refiere al servidor WSGI (Web Server Gateway Interface). En este proyecto este fichero no llega a usarse, pero se utiliza para desplegar nuestras aplicaciones en servidores como Apache. Pudiendo resultar de utilidad para la elaboración de ampliaciones dentro de la aplicación. En este fichero tampoco se deberán aplicar cambios.

asgi.py

Sirviendo como sucesor de WSGI, es la interfaz que se encuentra en las nuevas versiones de Django. Es otro fichero en el que no se deberán aplicar cambios.

views.py

Este fichero es uno de los más relevantes, ya que cuenta con todas las vistas con las que interactúa el usuario. Estas gestionan de manera superficial, sin entrar en gestiones de datos complejas las acciones del usuario. En este proyecto se cuenta con la vista `compilador`, la cual gestiona todas las peticiones referentes a la interacción con el compilador creado para el proyecto.

models.py

El fichero que contiene los `models` de la aplicación web, generalmente como clases. Suele ser los encargados de tratar datos de manera más exhaustiva y en este caso ejecuta de manera directa los comandos referentes al compilador, tratando sus entradas y salidas.

forms.py

Cuando se utilizan formularios HTML, es necesaria la comprobación de la validez de los campos introducidos por el usuario. De eso se encarga este fichero, que comprueba que el cuadro de texto correspondiente al compilador no esté vacío y sea válido.

templates

Es el directorio donde se guardan las plantillas correspondientes a las pantallas de la web. Este directorio debe indicarse en `settings.py` en la variable `TEMPLATES`, para que Django sepa de donde sacar las plantillas.

`static`

En Django se utiliza el directorio `static` para almacenar y organizar los archivos que no se modificarán, como `Javascripts`, `CSS`, imágenes y audios. En este caso se guardan los ficheros correspondientes a la compilación.

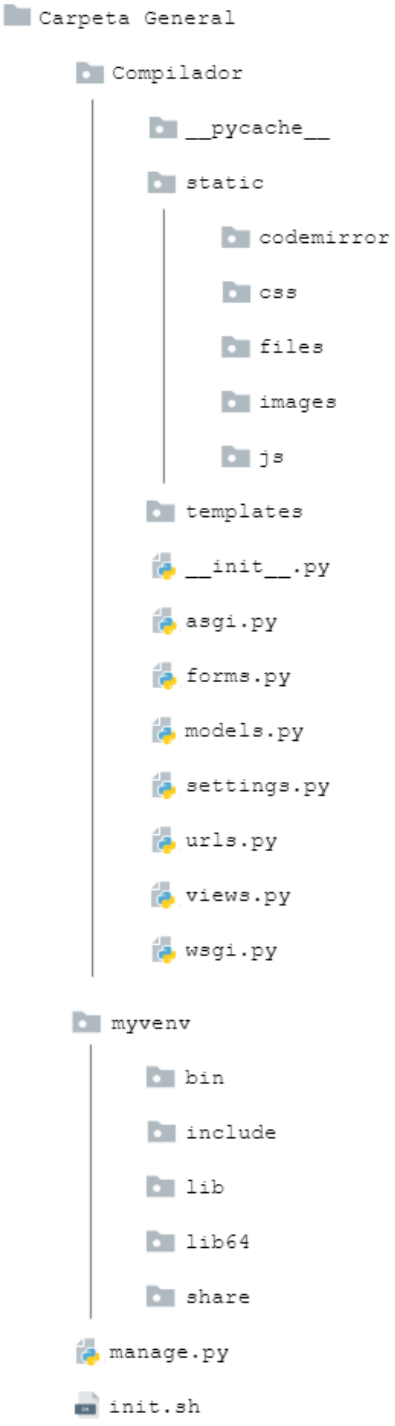


Figura 6.1: Estructura de ficheros de Django

6.3. Ejecución y uso

Una vez la implementación se ha llevado de manera correcta, es posible ejecutar el proyecto para usar la aplicación web desarrollada en Django. Para ello se pondrá en marcha la web con el siguiente comando, llevando acabo un despliegue en modo `DEBUG`, ya que por ahora el proyecto no esta en fase de producción:

```
python manage.py runserver
```

Ya puesto en marcha, podremos acceder a la URL indicada en el fichero `urls.py` que hace referencia a la vista con la plantilla correspondiente del compilador. En este caso es la IP de localhost `127.0.0.1:8000/compilador/`. Una vez se acceda aparecerá la pantalla con el cuadro de texto donde poder empezar a escribir las partituras en el lenguaje intermedio, la barra de navegación donde están los botones de las funcionalidades y el margen donde aparecerá la partitura una vez compilada, ver Figura 6.2.

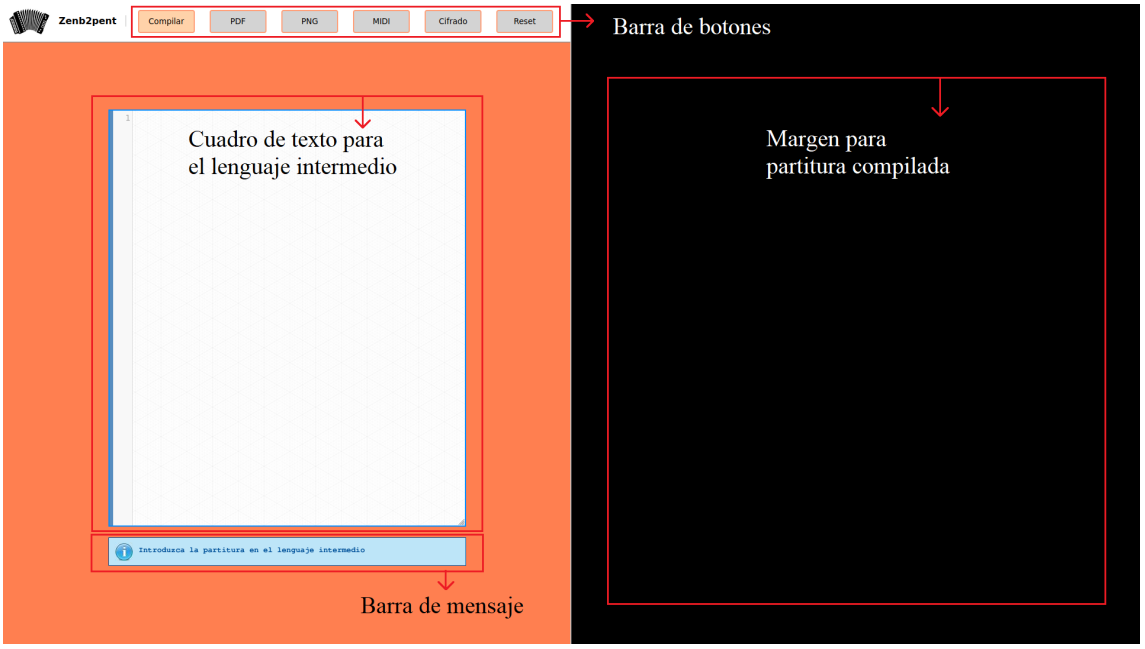


Figura 6.2: Aplicación web pre-compilado

Zenb2pent (zenbakiz to pentagrama), que es como se ha decidido bautizar a la aplicación, ofrece diversas funciones al usuario, a continuación se enumerarán para su mejor comprensión.

- **Compilar**: Si se pulsa este botón una vez introducida una partitura en lenguaje intermedio en el cuadro de texto de la margen izquierda, se obtendrá un mensaje indicando si la compilación ha sido correcta, o si se ha encontrado algún error. Si ocurre este

último, se indicará el primer error encontrado en la partitura. Cuando la compilación sea correcta, la partitura resultante se mostrará en la margen derecha (Figura 6.3).

- **PDF**: Si se pulsa este botón antes de introducir una partitura en el lenguaje intermedio, se avisará con un mensaje de que es necesario compilar una partitura previamente. Una vez compilada la partitura, podrá clicarse para abrir una pestaña con el fichero PDF correspondiente.
- **PNG**: Si se pulsa este botón antes de introducir una partitura en el lenguaje intermedio, se avisará con un mensaje de que es necesario compilar una partitura previamente. Una vez compilada la partitura, podrá clicarse para abrir una pestaña con la imagen PNG correspondiente.
- **MIDI**: Si se pulsa este botón antes de introducir una partitura en el lenguaje intermedio, se avisará con un mensaje de que es necesario compilar una partitura previamente. Una vez compilada la partitura, podrá clicarse para abrir una pestaña con la reproducción del audio MIDI correspondiente.
- **Cifrado**: Si se pulsa este botón antes de introducir una partitura en el lenguaje intermedio, se avisará con un mensaje de que es necesario compilar una partitura previamente. Una vez compilada la partitura, añadirá a la misma el cifrado americano debajo de las notas del bajo.
- **Reset**: Este botón sirve para borrar el caché generado durante la compilación, pudiendo volver al estado inicial de la web.
- **Mensajes**: El recuadro inferior al cuadro de texto, avisará de distinta manera si la compilación ha sido correcta, si han surgido errores (Figura 6.4), si es necesario realizar algún paso previo a la función seleccionada o simplemente indicando que se introduzca una partitura válida.
- **Cuadro de texto**: Es el cuadro de texto donde se escribe el lenguaje intermedio a compilar. Colorea de distinta manera las distintas partes del léxico y gramática de el lenguaje para mayor claridad del usuario.

Añadir que el repositorio donde se puede acceder a esta parte del proyecto es accesible a través de GitHub [1].

[1] https://github.com/AlvaroLuzu/django_TFG

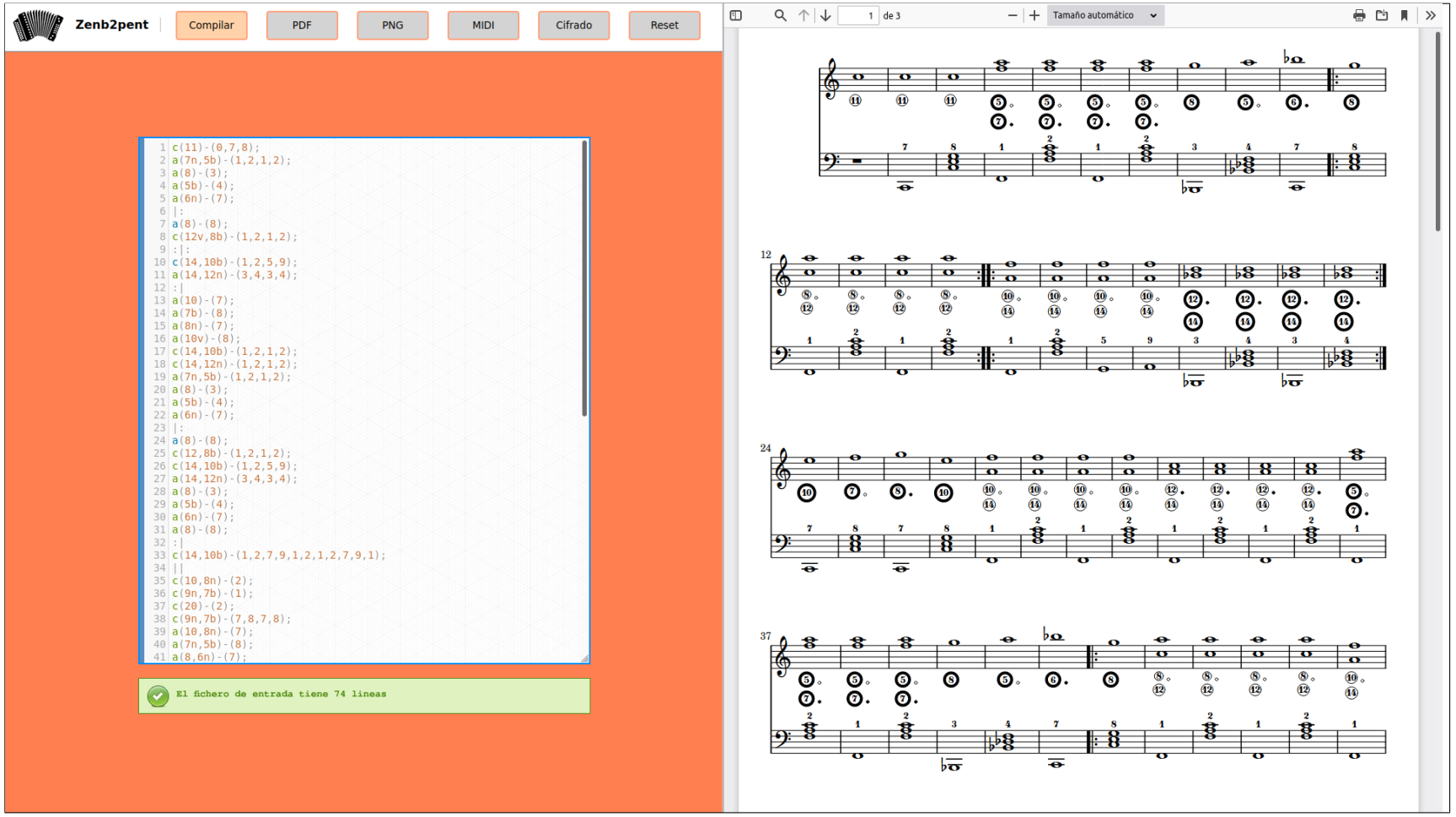


Figura 6.3: Aplicación web post-compilado

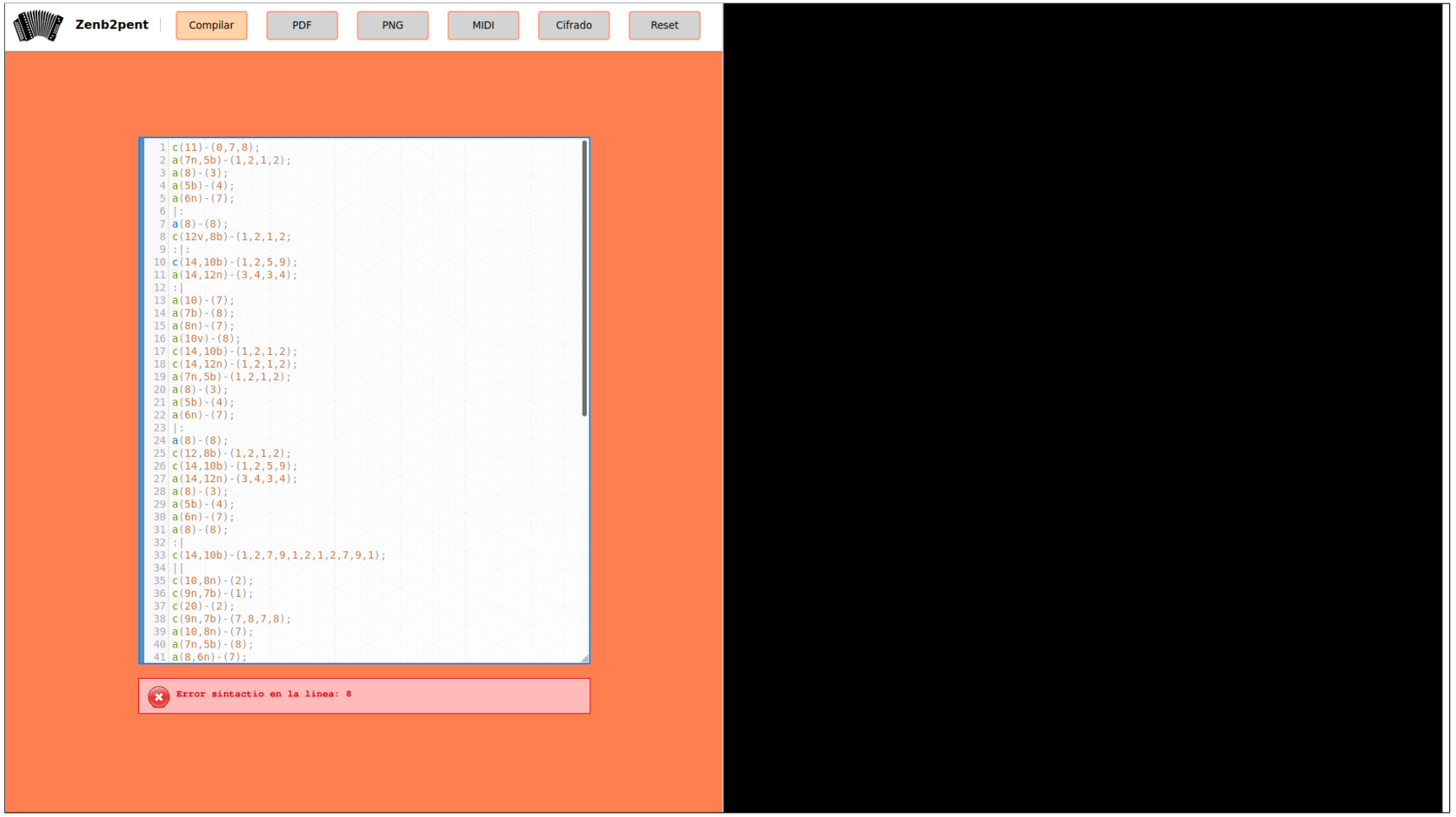


Figura 6.4: Aplicación web error de compilado

CAPÍTULO 7

Pruebas de software

En el desarrollo de todo tipo de software, las pruebas son un apartado de gran importancia. Sin un buen plan de pruebas es posible que el trabajo desarrollado no contemple casos de prueba que se le requerían desde un inicio. Esto es dado a no haberlos podido considerar durante la implementación del propio software, ya que se puede llegar a olvidar alguna situación en la que el código debería funcionar y no es visible hasta que se comprueba en la versión final [18]. Para poder suplir estos posibles problemas se han desarrollado las siguientes pruebas de software.

7.1. Compilación y creación de partitura

Id. de prueba	Descripción	Resultado esperado	Resultado obtenido	Observaciones
1	Crear partitura vacía, el fichero de entrada estará vació	Una partitura únicamente con claves vacías	Una partitura únicamente con claves vacías	Correcto
2.1	Crear partitura con solo una nota simultánea en la melodía, sin digitación, fuelle abriendo y un compás	Partitura con las notas y escritura de trikitixa correspondientes. Un compás en clave de Sol, y silencios en el de Fa	Partitura con las notas y escritura de trikitixa correspondientes. Un compás en clave de Sol, y silencios en el de Fa	Correcto
2.1.1	Crear partitura con solo una nota simultánea en la melodía, sin digitación, fuelle cerrando y un compás	Partitura con las notas y escritura de trikitixa correspondientes. Un compás en clave de Sol, y silencios en el de Fa	Partitura con las notas y escritura de trikitixa correspondientes. Un compás en clave de Sol, y silencios en el de Fa	Correcto
2.1.2	Crear partitura con varias notas simultáneas en la melodía, sin digitación y un compás	Partitura con la digitación correspondiente. Un compás en clave de Sol, y silencios en el de Fa	Partitura con la digitación correspondiente. Un compás en clave de Sol, y silencios en el de Fa	Correcto

Tabla 7.1: Compilación y creación de partitura (1)

2.1.3	Crear partitura con solo melodía, con digitación y un compás	Partitura con la digitación correspondiente. Un compás en clave de Sol, y silencios en el de Fa	Partitura con la digitación correspondiente. Un compás en clave de Sol, y silencios en el de Fa	Correcto
2.1.4	Crear partitura con un sonido bajo y un compás	Partitura con las notas y escritura de trikitixa correspondientes. Silencios en clave de Sol, y un compás en el de Fa	Partitura con las notas y escritura de trikitixa correspondientes. Silencios en clave de Sol, y un compás en el de Fa	Correcto
2.1.5	Crear partitura con varios sonidos simultáneos en el bajo y un compás	Partitura con las notas y escritura de trikitixa correspondientes. Silencios en clave de Sol, y un compás en el de Fa	Partitura con las notas y escritura de trikitixa correspondientes. Silencios en clave de Sol, y un compás en el de Fa	Correcto
2.1.6	Crear partitura con varios sonidos no simultáneos en el bajo y un compás	Partitura con las notas y escritura de trikitixa correspondientes. Silencios en clave de Sol, y un compás en el de Fa	Partitura con las notas y escritura de trikitixa correspondientes. Silencios en clave de Sol, y un compás en el de Fa	Correcto

Tabla 7.2: Compilación y creación de partitura (2)

2.1.7	Crear partitura con melodía y bajo en un compás	Partitura con las notas y escritura de trikitixa correspondientes. Un compás con notas en clave de Sol y en clave de Fa	Partitura con las notas y escritura de trikitixa correspondientes. Un compás con notas en clave de Sol y en clave de Fa	Correcto
2.2	Crear partitura con solo melodía, más de un compás	Partitura con varios compases en clave de Sol, y silencios en el de Fa	Partitura con varios compases en clave de Sol, y silencios en el de Fa	Correcto
2.2.1	Crear partitura con solo bajo, más de un compás	Partitura con silencios en clave de Sol, y varios compases con notas en el de Fa	Partitura con silencios en clave de Sol, y varios compases con notas en el de Fa	Correcto
2.2.2	Crear partitura con bajo y melodía, más de un compás	Partitura con varios compases tanto en clave de Sol como en clave de Fa	Partitura con varios compases tanto en clave de Sol como en clave de Fa	Correcto
2.2.3	Crear partitura con bajo y melodía, más de un compás, con comas en el bajo para la repetición de la melodía	Partitura con varios compases tanto en clave de Sol como en clave de Fa, melodía repetida el número de veces de comas menos uno	Partitura con varios compases tanto en clave de Sol como en clave de Fa, melodía repetida el número de veces de comas menos uno	Correcto

Tabla 7.3: Compilación y creación de partitura (3)

2.3	Crear partitura con silencios en melodía y bajo, un compás	Partitura con silencios tanto en la melodía como en el bajo	Partitura con silencios tanto en la melodía como en el bajo	Correcto
2.3.1	Crear partitura con silencios en melodía y bajo, varios compases	Partitura con silencios tanto en la melodía como en el bajo	Partitura con silencios tanto en la melodía como en el bajo	Correcto
2.4	Crear partitura con melodía o bajo, faltando algún símbolo como `(`, `a / c`, `-` para el bajo etc.	Partitura correcta hasta el compás del error encontrado	Partitura correcta hasta el compás del error encontrado	La gestión de errores no contempla completamente los caracteres o símbolos restantes
3	Crear partitura con una melodía en tresillo, con tres sonidos entre corchetes, sin bajo	Partitura con el tresillo correspondiente	Partitura con el tresillo correspondiente	Correcto
3.1	Crear partitura con una melodía en tresillo, con tres sonidos entre corchetes, con bajo	Partitura con el tresillo correspondiente	Partitura con el tresillo correspondiente	Correcto

Tabla 7.4: Compilación y creación de partitura (4)

3.2	Crear partitura con una melodía en tresillo, con menos tres sonidos entre corchetes	Partitura errónea, con fallos	Partitura errónea, con fallos	La notación de tresillos debe ser la correcta sino las variables correspondientes a los pentagramas fallarán
4	Crear partitura únicamente símbolos de repetición	Partitura vacía	Partitura vacía	Musicalmente no tiene sentido crear una partitura con símbolos de repetición si no existen sonidos en ella
4.1	Crear partitura con símbolos de repetición y sonidos	Partitura correcta, sin fallos	Partitura correcta, sin fallos	Correcto
5	Crear partitura con símbolos extraños o repeticiones innecesarias	Partitura correcta hasta el primer símbolo extraño. El compilador indica la línea donde se encuentra el error	Partitura correcta hasta el primer símbolo extraño. El compilador indica la línea donde se encuentra el error	El compilador indica la línea y generalmente el símbolo esperado donde está el erróneo

Tabla 7.5: Compilación y creación de partitura (5)

7.2. Funcionalidades de la Web

Id. de prueba	Descripción	Resultado esperado	Resultado obtenido	Observaciones
1	Clic en el botón 'Compilar' con el cuadro de texto vacío	Se muestra una partitura vacía	Partitura vacía	Correcto
1.1	Clic en el botón 'PDF' sin haber compilado previamente una partitura	Mensaje avisando de la falta de una partitura a introducir	Mensaje avisando de la falta de una partitura a introducir	Correcto
1.2	Clic en el botón 'PNG' sin haber compilado previamente una partitura	Mensaje avisando de la falta de una partitura a introducir	Mensaje avisando de la falta de una partitura a introducir	Correcto
1.3	Clic en el botón 'MIDI' sin haber compilado previamente una partitura	Mensaje avisando de la falta de una partitura a introducir	Mensaje avisando de la falta de una partitura a introducir	Correcto
1.4	Clic en el botón 'Cifrado' sin haber compilado previamente una partitura	Mensaje avisando de la falta de una partitura a introducir	Mensaje avisando de la falta de una partitura a introducir	Correcto
1.5	Clic en el botón 'Reset' sin haber compilado previamente una partitura	Mensaje informando para que se introduzca una partitura	Mensaje informando para que se introduzca una partitura	Correcto

Tabla 7.6: Funcionalidades de la Web (1)

Id. de prueba	Descripción	Resultado esperado	Resultado obtenido	Observaciones
2	Clic en el botón 'Compilar' con una partitura sin errores en el cuadro de texto	Se muestra la partitura completa y mensaje de compilación correcta junto al número de líneas	Se muestra la partitura completa y mensaje de compilación correcta junto al número de líneas	Correcto
2.1	Clic en el botón 'PDF' habiendo compilado una partitura sin errores	Se abre una pestaña con el PDF correspondiente a la partitura	Se abre una pestaña con el PDF correspondiente a la partitura	Correcto
2.2	Clic en el botón 'PNG' habiendo compilado una partitura sin errores	Se abre una pestaña con el PNG correspondiente a la partitura	Se abre una pestaña con el PNG correspondiente a la partitura	Correcto
2.3	Clic en el botón 'MIDI' habiendo compilado una partitura sin errores	Se abre una pestaña con el MIDI correspondiente a la partitura	Se abre una pestaña con el MIDI correspondiente a la partitura	Correcto
2.4	Clic en el botón 'Cifrado' habiendo compilado una partitura sin errores	Se muestra la partitura completa con el cifrado americano con mensaje de compilación correcta junto al número de líneas	Se muestra la partitura completa con el cifrado americano con mensaje de compilación correcta junto al número de líneas	Correcto

Tabla 7.7: Funcionalidades de la Web (2)

Id. de prueba	Descripción	Resultado esperado	Resultado obtenido	Observaciones
2.5	Clic en el botón 'Reset' habiendo compilado una partitura sin errores	Se eliminan los archivos generados con la compilación, quitando la partitura visible y limpiando el cuadro de texto	Se eliminan los archivos generados con la compilación, quitando la partitura visible y limpiando el cuadro de texto	Correcto
3.0	Clic en el botón 'Compilar' con una partitura con un error en la primera linea del cuadro de texto	Muestra un mensaje de error indicando el número de la línea con el error	Muestra un mensaje de error indicando el número de la línea con el error	Correcto
3.0.1	Clic en el botón 'Compilar' con una partitura con un error medio del cuadro de texto	Muestra un mensaje de error indicando el número de la línea con el error	Muestra un mensaje de error indicando el número de la línea con el error	Correcto
3.0.2	Clic en el botón 'Compilar' con una partitura con un error en el final del cuadro de texto	Muestra un mensaje de error indicando el número de la línea con el error	Muestra un mensaje de error indicando el número de la línea con el error	Correcto
3.0.3	Clic en el botón 'Compilar' con una partitura con varios errores en el cuadro de texto	Muestra un mensaje de error indicando el número de la línea con el primer error	Muestra un mensaje de error indicando el número de la línea con el primer error	Correcto

Tabla 7.8: Funcionalidades de la Web (3)

Id. de prueba	Descripción	Resultado esperado	Resultado obtenido	Observaciones
3.1	Clic en el botón 'PDF' habiendo compilado una partitura con errores	Muestra la partitura previa a la compilación con errores	Muestra la partitura previa a la compilación con errores	Correcto
3.2	Clic en el botón 'PNG' habiendo compilado una partitura con errores	Muestra la partitura previa a la compilación con errores	Muestra la partitura previa a la compilación con errores	Correcto
3.3	Clic en el botón 'MIDI' habiendo compilado una partitura con errores	Muestra el MIDI previo a la compilación con errores	Muestra el MIDI previo a la compilación con errores	Correcto
3.4	Clic en el botón 'Cifrado' habiendo compilado una partitura con errores	Muestra la partitura previa a la compilación con errores	Muestra la partitura previa a la compilación con errores	Correcto
3.5	Clic en el botón 'Reset' habiendo compilado una partitura con errores	Se eliminan los archivos generados con la compilación, quitando la partitura visible y limpiando el cuadro de texto	Se eliminan los archivos generados con la compilación, quitando la partitura visible y limpiando el cuadro de texto	Correcto

Tabla 7.9: Funcionalidades de la Web (4)

CAPÍTULO 8

Conclusiones

Llegando al final de esta memoria podemos concluir diciendo que el trabajo resultante es que el que esperábamos. Ya que se ha confeccionado un compilador con Flex y Bison, capaz de procesar un nuevo lenguaje semejante a lo que sería pasar a texto plano la escritura utilizada por los trikitilaris, a una notación (Lilypond) con la que somos capaces de obtener su equivalente en partitura universal. Además se ofrece este compilador a los usuarios que lo necesiten a través de un sitio web, gracias a haber desarrollado una aplicación web que lo contiene hecha con el framework para desarrollo de páginas web Django.

Además de que es la única herramienta existente que suple el problema que surge entre trikitilaris y otros músicos a la hora de comunicarse por vía escrita. Dado a que los primeros utilizan una notación vasca para escribir música de acordeón diatónico en papel, el cual no es conocido para el resto de músicos habituados al pentagrama universal. Esta falta de comunicación entorpece la colaboración entre grupos de músicos de ambas especialidades. Ofreciendo así una solución en forma de partitura con la que ambos tipos de intérpretes podrían tocar sus obras, ya que el resultado no solo es la transformación de la escritura 'zenbakizko', sino que esta escritura junto con el pentagrama universal son mostrados una al lado de la otra.

El proyecto finalmente ha cumplido con los objetivos principales mencionados en el Capítulo 3:

- Definir un lenguaje intermedio para la escritura de acordeón ✓
- Crear un compilador para convertir el lenguaje intermedio a Lilypond, desde la cual se obtendrá el archivo PDF. El compilador tendrá en cuenta las siguientes notaciones:
    - Melodía en clave de Sol ✓

        - Bajo en clave de Fa ✓
        - Notas y Acordes de distinta altura ✓
        - Alteraciones ✓
        - Digitación y escritura numérica de acordeón ✓
    - Mostrar la escritura de trikitixa junto a la partitura compilada ✓
    - Página web donde poder realizar las siguientes funciones:
        - Importar un archivo de texto en lenguaje intermedio y exportarlo a PDF ✓
        - Escribir en la propia web el lenguaje intermedio para poder compilar y obtener una previsualización ✓

Y la mayoría de los objetivos secundarios:

- Implementar la exportación de lenguaje intermedio a MIDI ✓
- Mostrar el resultado en MIDI en la web al compilar ✓
- Añadir siguientes notaciones al lenguaje intermedio y por tanto a la partitura:
    - Silencios ✓
    - Marcas de compás y repeticiones ✓
    - Definición de compás ✗
    - Notas con medidas y valores irregulares ✓
    - Notas con puntillo ✓
    - Indicadores de expresión (ligaduras, *staccato*...) ✗
    - Intensidad (forte, piano...) ✗
    - Tempo (*adagio*, *allegro*...) ✗
- Implementar el cifrado americano para los acordes según cambian en la partitura ✓
- Seguimiento de la partitura mientras se reproduce el MIDI en la web ✗

Los objetivos no cumplidos podemos incluirlos en el trabajo a futuro, ya que principalmente estas finalidades dependerían de otro tipo de trabajo para ser elaboradas. Véase los indicadores de compás, expresión, intensidad y tempo que no aparecen representados en una escritura de trikitixa, pero se podría desarrollar una Inteligencia Artificial capaz de analizar una partitura de trikitixa comparándola con una base de datos de partituras similares donde ya se conocería si estos resultados son esperados en un pentagrama o no. También sería posible modelar la digitación utilizando una base de datos y añadirle al proyecto un sistema inteligente que asigne la digitación/fuelle automáticamente a una partitura al cual no se le haya indicado esta información previamente. Aunque las bases de datos en el contexto de acordeón diatónico son pequeños, estos modelos de digitación se podrían pre-entrenar utilizando el aprendizaje por transferencia ya que hay bases de datos masivas de digitación en el contexto del instrumento piano. Mencionar que la reproducción de audio realice un seguimiento nota por nota por la partitura compilada, conllevaría a utilizar

herramientas no pensadas para este proyecto, ya que PDF y PNG no ofrecen alternativas para ello.

Gracias al aprendizaje de las herramientas utilizadas, se puede ver como elaborar mejoras o modificaciones como trabajo futuro para la ampliación de este proyecto. Una de las ideas era implementar una funcionalidad en la web, donde insertar un enlace de una partitura en Trikitirauki[1], y mediante un parser en Python analizar el contenido HTML para así generar el lenguaje intermedio y obtener la partitura resultante. Otra es poder importar otros formatos musicales como Lilypond, MusicXML y que el compilador realice el proceso inverso, transformándolos al lenguaje intermedio. Finalmente otra mejora a implementar un sistema de gestión de usuarios en la web, donde cada usuario podrá crear y gestionar diversas partituras desde un perfil.

Mencionar que el esfuerzo inicialmente propuesto ha supuesto una estimación medianamente correcta de las horas invertidas. El hecho de haber tenido que aprender a utilizar herramientas nuevas con las que trabajar, ha supuesto una curva de aprendizaje algo pronunciada, por lo que esas horas se han visto incrementadas considerablemente. Sumar esto a posibles retrasos ha hecho que haya habido que dedicarle un tiempo extra al proyecto, sin desviarse del plan original, ya que este problema se ha suplido con más cantidad de horas invertidas al día.

Para concluir me gustaría exponer mi experiencia con este proyecto. Desde que comenzó la carrera, el tema del TFG era algo que me surgían muchas dudas e inquietudes. Siempre pensaba que llegaría el día en tener que hacerlo y no saber qué hacer, o tener que realizar un trabajo de tanto esfuerzo que no me motivase para nada. A veces pensaba de manera algo fantasiosa enlazar mi pasión musical, a la que había dedicado tantas horas de mi vida con mis conocimientos dentro del grado. Pero estas ideas se iban rápidamente de mi cabeza ya que veía imposible realizar una tarea así por la complejidad que supondría, o porque ya se habían tocado todos los palos posibles.

Cuando llegó el curso donde se debía comenzar con el TFG tocaba elegir definitivamente un tema, así que me acerqué a los temas ofertados por los profesores para ver si me inspiraban a encontrar algo que me satisficiese. Vi entre todos los temas ofertados el ofrecido por Iñigo Perona e Iñigo Mendialdua, profesores con los que ya había tratado y con gusto realizaría un trabajo.

Así que así fue como comenzó este proyecto. Uno que ha conllevado a aprender a utilizar muchas herramientas nuevas y a supuesto una curva de aprendizaje ligeramente elevada. La elección de usar estas herramientas fue el fruto de tiempo de investigación, ya que existía un desconocimiento sobre la edición de notación musical en la informática, la creación de compiladores y el desarrollo de páginas web de manera sencilla. Aunque el tiempo invertido ha merecido la pena, ya se ha visto que dichas herramientas han cumplido con creces sus funciones. Flex y Bison son capaces

[1] https://tirikitrauki.com/zenbakiz/

de ofrecer analizadores léxicos y sintácticos de cualquier tipo si se conoce bien como sacar partido a las capacidades del LR Parser, permitiendo así definir el lenguaje requerido. Lilypond es una notación con muchas opciones y profundidad a la hora de la creación de partituras musicales. Y Django se ha presentado como una de las mejores opciones para el desarrollo web sencillo pero con suficiente complejidad.

Aunque la gestión y planificación no ha sido incorrecta, es cierto que unas mejoras en esta y en el tiempo invertido podría haber supuesto un aprendizaje más rápido de las herramientas utilizadas, lo que hubiera supuesto la capacidad por nuestra parte en un mayor rango de funcionalidades implementadas. Sin embargo, el trabajo realizado suple un problema real existente en el mundo de la música, más concretamente aquí en el País Vasco. Siendo finalmente una herramienta única por los ámbitos en los que se mueve, el problema al que pone solución y el como intenta solventarlo.

Bibliografía

[1] Euskal Herriko Trikitixa Elkartea. Zer da trikitixa? https://trikitixa.eus/zer-da-trikitixa/.

[2] Pablo Domínguez. En qué consiste el modelo en cascada. https://openclassrooms.com/en/courses/4309151-gestiona-tu-proyecto-de-desarrollo/4538221-en-que-consiste-el-modelo-en-cascada, 5 2021.

[3] Agencia Estatal Boletín Oficial del Estado. BoletÍn oficial del estado - iii. otras disposiciones - ministerio de trabajo y economÍa social. https://www.boe.es/boe/dias/2020/05/13/pdfs/BOE-A-2020-5006.pdf, 5 2020.

[4] Ada Funes. Irpf 2021: cómo calcular cuánto me tienen que retener. https://www.elespanol.com/como/calcular-irpf-retener/455704557_0.html, 1 2021.

[5] Aitor Valle Allende. Xmlscore: Representación gráfica y reproducción de partituras en formato xml. https://addi.ehu.es/bitstream/handle/10810/18388/avalle004_XMLScore_Kosmos_memoria.pdf?sequence=4&isAllowed=y, 2 2016.

[6] Tec de Monterrey Ramon Brena. *Automatas y Lenguajes - Un enfoque de diseño*. Editorial McGraw-Hill, 2003. Disponible en http://ciencias.uis.edu.co/lenguajes/doc/Automatas%20y%20Lenguajes%20-%20Ramon%20Brena.pdf.

[7] Ravi Sethi y Jeffrey D. Ullman Alfred V. Aho, Monica S. Lam. *Compiladores: principios, técnicas y herramientas*. Greg Tobin, 1986.

[8] Boštjan Slivnik. LLLR Parsing: a Combination of LL and LR Parsing. *SLATE*, 16:Articulo Nº5, páginas 1–13, 2016. Disponible en https://api.semanticscholar.org/CorpusID:1965041.

[9] Kenneth C. Louden. *Construcción de compiladores - Principios y práctica.* International Thomson Editores, 2004.

[10] Pankaj Patel. Bundelkhand Institute of Engineering & Technology (BIET) Jhansi. Difference between LL and LR parser. https://www.geeksforgeeks.org/difference-between-ll-and-lr-parser/, Julio 2019.

[11] Vern Paxson. *Flex, versión 2.5 - Un generador de analizadores léxicos rápidos.* Universidad de California, 1995. Disponible en http://webdiis.unizar.es/asignaturas/LGA/docs_externos/flex-es-2.5.pdf.

[12] Federico Simmross Wattenberg. El generador de analizadores léxicos lex. https://www.infor.uva.es/~mluisa/talf/docs/labo/L3.pdf.

[13] Anthony A. Aaby. *Compiler Construction using Flex and Bison.* Walla Walla College, 2003. Disponible en https://dlsiisv.fi.upm.es/traductores/Software/Flex-Bison.pdf.

[14] Federico Simmross Wattenberg. El generador de analizadores sintácticos yacc. https://www.infor.uva.es/~mluisa/talf/docs/labo/L8.pdf.

[15] Vern Paxson. *Bison - El Generador de Analizadores Sintácticos compatible con YACC.* Free Software Foundation, 1995. Disponible en http://webdiis.unizar.es/asignaturas/LGA/docs_externos/bison-es-1.27.pdf.

[16] Rubén Béjar Hernández. Dpto. Informática e Ingeniería de Sistemas Universidad de Zaragoza. Introducción a flex y bison. http://webdiis.unizar.es/asignaturas/LGA/material_2004_2005/Intro_Flex_Bison.pdf, 2004-2005.

[17] Documentación de django. https://docs.djangoproject.com/en/3.2/.

[18] Gaurav Dubey Maneela Tuteja. A research study on importance of testing and quality assurance in software development life cycle (sdlc) models. *International Journal of Soft Computing and Engineering (IJSCE)*, 2:251–257, 2012. Disponible en https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.459.5873&rep=rep1&type=pdf.

[19] Manuales de lilypond 2.22.1. http://lilypond.org/manuals.es.html.

APÉNDICE A

Escritura de trikitixa

A continuación se darán unas nociones sobre la escritura de trikitixa y como estas se representan en una partitura.

Los números dentro de los círculos corresponden a los botones a tocar en un acordeón con la mano derecha. El acordeón diatónico o trikitixa cuenta con 23 botones en este lado. Estas notas van vinculadas a la melodía como se detalla en la Sección 5.1. El estado del fuelle hace variar el sonido al pulsar estos botones, las notas correspondientes a los números mientras el fuelle está abriéndose corresponden a la siguiente Figura A.1

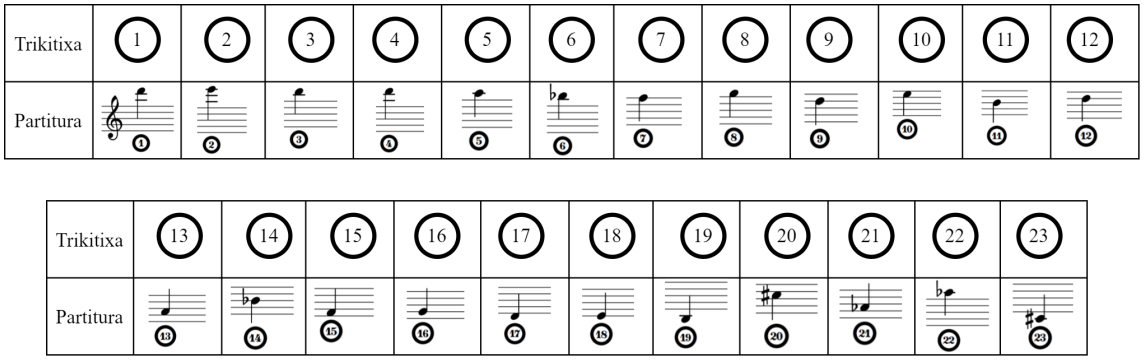


Figura A.1: Notas de la melodía, fuelle abriendo

Las notas correspondientes a los números mientras el fuelle está cerrándose corresponden a la siguiente Figura A.2

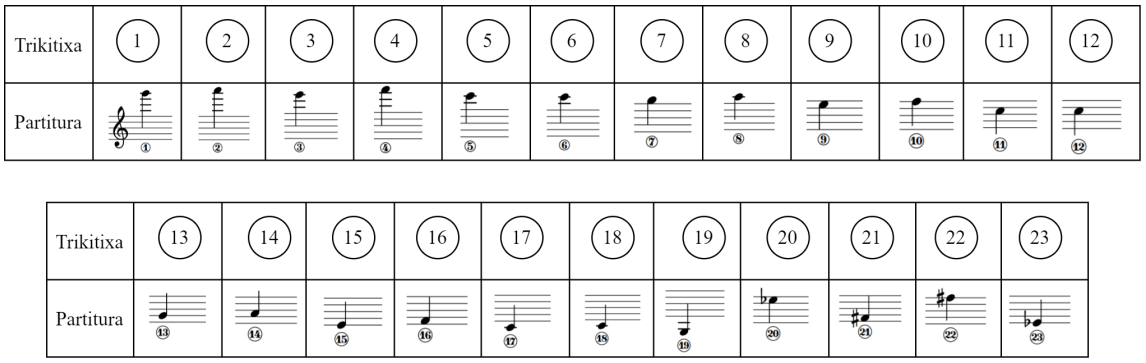


Figura A.2: Notas de la melodía, fuelle cerrando

La digitación es una característica de la escritura de trikitixa que no tiene relevancia en la partitura resultante, ya que es una indicación orientada a la técnica, no al resultado musical. Aún así este sería el resultado en Lilypond añadiendo la digitación dentro de la partitura (Figura A.3):

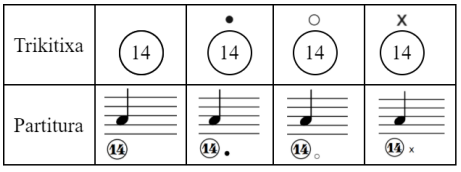


Figura A.3: Melodía con digitacion

En cuanto a los números colocados debajo de los círculos correspondientes a la melodía, pertenecen al bajo. Estos son 12 botones que a diferencia de la melodía no van vinculados al estado del fuelle, y sus notas correspondientes son las siguientes (Figura A.4):

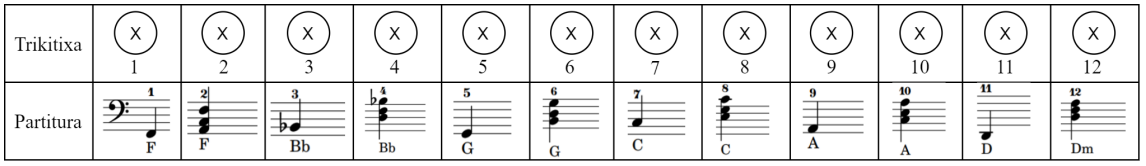


Figura A.4: Notas del bajo

Los círculos de la melodía pueden ir tanto separados indicando su no simultaneidad sonora en el pulso, como juntas indicando que si sonarán de manera simultánea (Figura A.5):

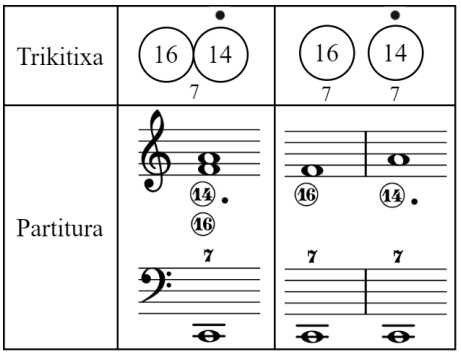


Figura A.5: Melodía de varias notas, juntas y separadas

Es posible expresar distintos ritmos o repeticiones con símbolos distintivos en el bajo. Entre ellos se usan:

- Los espacios en blanco entre bajos, para denotar que ambos números contiguos se tocarán de manera progresiva.
- La coma, para indicar que lo siguiente a ella deberá sonar con una nueva pulsación de la melodía a la que corresponde ese bajo. Esta coma puede venir junto un número encima de las notas de la melodía para indicar el número de repeticiones.
- El bajo con circunferencia, dos números unidos cada uno con una circunferencia, indicando que son sonidos que sonarán simultáneamente.

Estos ejemplos son visibles en la Figura A.6:

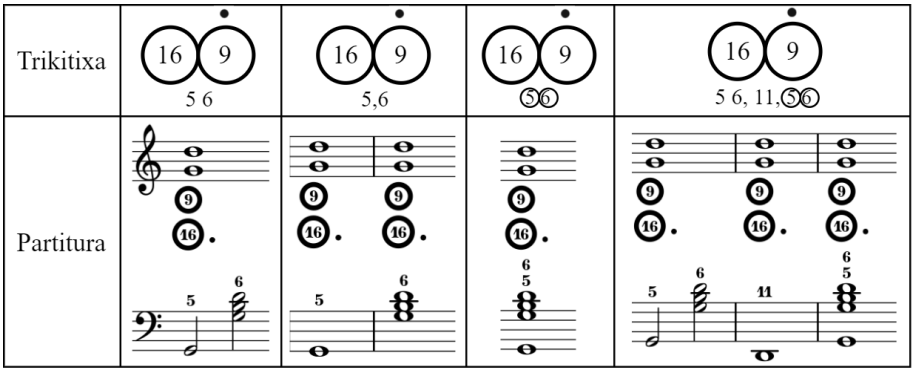


Figura A.6: Distintos símbolos en el bajo

Los silencios musicales también tiene representación dentro de la escritura de la trikitixa. Se representan como puntos negros y pueden usarse tanto en la melodía como en el bajo (Figura A.7).

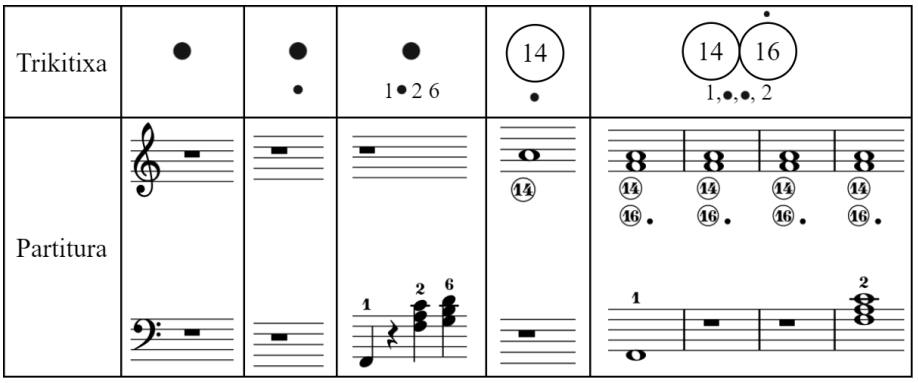


Figura A.7: Silencios en melodía y bajo

Se encuentran también tresillos, incluyendo tres notas no contiguas de la melodía, envueltos por un corchete horizontal. Siempre viene acompañado de un único bajo. Es posible apreciarlo mejor en la Figura A.8.

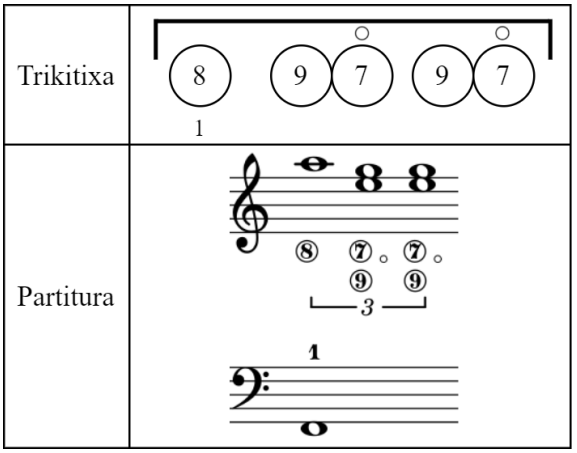


Figura A.8: Tresillos en la melodía

A la hora de representar con coda, se usa un símbolo de ‘estrella’ o asterisco para denotar de donde a donde se debe saltar en la interpretación (Figura A.9).

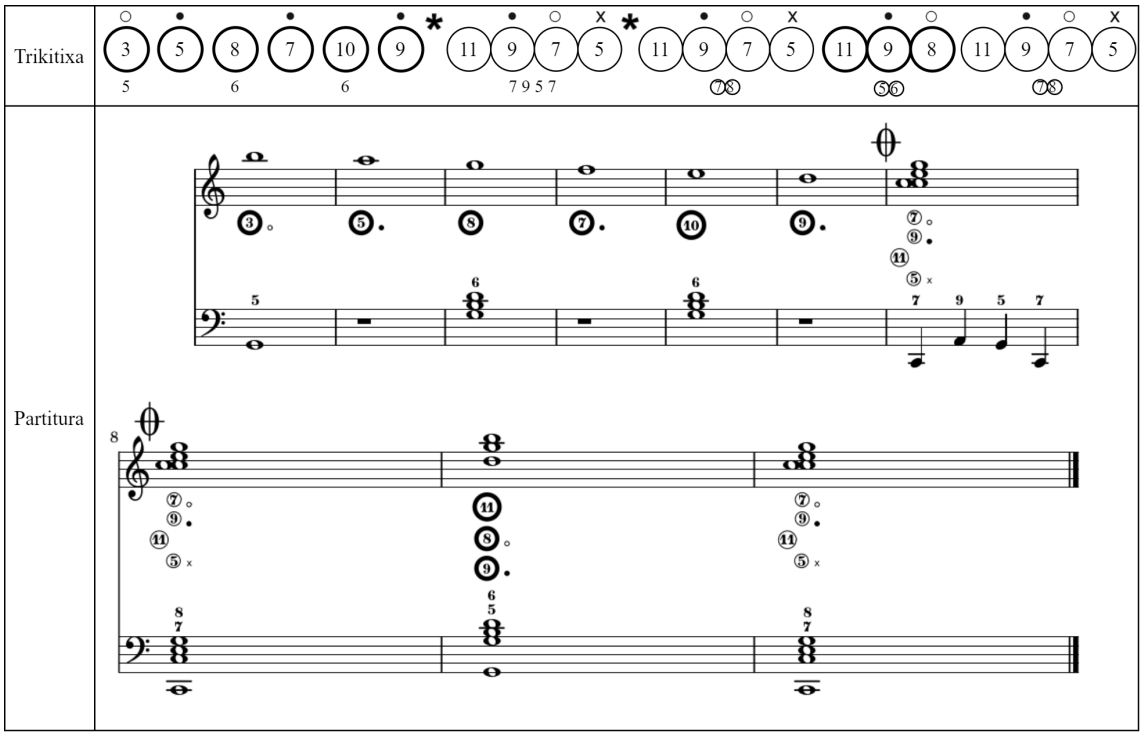


Figura A.9: Fragmento con representación de una coda

APÉNDICE B

LILYPOND

Utilizando la completa y extensa guía de Lilypond [19], ha sido posible la realización de partituras que se puedan asemejar lo máximo posible a lo que se obtendría pasando a mano un escrito del lenguaje de trikitixa, a partitura de pentagrama clásico. Lilypond además potencia el apartado visual y de personalización de las partituras, buscando en lo máximo posible asemejarse a partituras realizadas de manera manual.

Los ficheros obtenidos al compilar el lenguaje intermedio de entrada tendrán una estructura y contenido similar a la siguiente:

```
1 \version "2.18.2"
2 penta_sol = {}
3 penta_fa = {}
4 penta_sol = { \penta_sol < b''\finger\markup{\override #'(thickness . 5) \circle
      {3} \draw-circle #0.25 #0 ##t} > 1 }
5 penta_fa = { \penta_fa \set fingeringOrientations = #'(up) <g,-5> 1 _\markup{ \
      abs-fontsize #10 G } }
6 penta_sol = { \penta_sol < > r 1 <> 1 }
7 penta_fa = { \penta_fa \set fingeringOrientations = #'(up) <bes,,-3 bes,-4 des f>
       1 _\markup{ \abs-fontsize #10 Bb } }
8 penta_fa = { \penta_fa \set fingeringOrientations = #'(up) <f,-1> 1 _\markup{ \
      abs-fontsize #10 F } }
```

```
9  penta_sol = { \penta_sol \tuplet 3/1 { < a''\finger\markup{\override #'(thickness
       . 5) \circle{5} " "} > < f''\finger\markup{\override #'(thickness . 5) \
       circle{7} " "} > < g''\finger\markup{\override #'(thickness . 5) \circle{8} "
       "} > } }
10 penta_sol = { \penta_sol \bar ":|." }
11 penta_fa = { \penta_fa \bar ":|." }
12 penta_sol = { \penta_sol < e''\finger\markup{\override #'(thickness . 1) \circle
       {9} " "} c''\finger\markup{\override #'(thickness . 1) \circle{12} \draw-
       circle #0.25 #0 ##t} > 1 }
13 penta_fa = { \penta_fa \set fingeringOrientations = #'(up) <g,-5> 2 _\markup{ \
       abs-fontsize #10 G } <c-8 e g> 2 _\markup{ \abs-fontsize #10 C } }
14 penta_sol = { \penta_sol < e''\finger\markup{\override #'(thickness . 1) \circle
       {9} " "} c''\finger\markup{\override #'(thickness . 1) \circle{12} \draw-
       circle #0.25 #0 ##t} > 1 }
15 penta_fa = { \penta_fa r }
16  penta_sol = { \penta_sol \once \override Score.RehearsalMark.font-size = #4 \
       mark \markup { \musicglyph #"scripts.coda" } }
17 penta_fa = { \penta_fa \once \override Score.RehearsalMark.font-size = #4 \mark \
       markup { \musicglyph #"scripts.coda" } }
18 penta_sol = { \penta_sol < e''\finger\markup{\override #'(thickness . 1) \circle
       {9} \draw-circle #0.25 #0 ##f} f'''\finger\markup{\override #'(thickness . 1)
       \circle{4} \abs-fontsize #3 x} > 1 }
19 penta_fa = { \penta_fa \set fingeringOrientations = #'(up)\tuplet 3/1 { <f,-1>_\
       markup{ \abs-fontsize #10 F } <f-2 a c'> <f,-1> } }
20 penta_sol = { \penta_sol \break }
21 penta_fa = { \penta_fa \break }
22  penta_sol = { \penta_sol \once \override Score.RehearsalMark.font-size = #4 \
       mark \markup { \musicglyph #"scripts.coda" } }
23 penta_fa = { \penta_fa \once \override Score.RehearsalMark.font-size = #4 \mark \
       markup { \musicglyph #"scripts.coda" } }
24 penta_sol = { \penta_sol < f'''\finger\markup{\override #'(thickness . 1) \circle
       {4} " "} c''\finger\markup{\override #'(thickness . 1) \circle{5} \draw-
       circle #0.25 #0 ##t} > 1 }
25 penta_sol = { \penta_sol < f'''\finger\markup{\override #'(thickness . 1) \circle
       {4} " "} c''\finger\markup{\override #'(thickness . 1) \circle{5} \draw-
       circle #0.25 #0 ##t} > 1 }
```

```
26 penta_fa = { \penta_fa \set fingeringOrientations = #'(up) <bes,-4 des f> 1 _\
       markup{ \abs-fontsize #10 Bb } <bes,-4 des f> 1 }
27 \version "2.18.2"
28
29 \paper {
30 system-system-spacing =
31 #'((basic-distance . 16)
32 (minimum-distance . 8)
33 (padding . 1)(stretchability . 60))
34 }
35
36 \score{
37 <<
38 \new Staff
39   \with { \remove "Time_signature_engraver" }{
40   \set fingeringOrientations = #'(down)
41   \clef "treble"
42   \penta_sol
43   \bar "|."
44 }
45 \new Staff
46   \with { \remove "Time_signature_engraver" }{
47   \set fingeringOrientations = #'(up)
48   \clef "bass"
49   \penta_fa
50   \bar "|."
51 }
52 >>
53 }
54 % El fichero de entrada tiene 14 lineas
```

Viendo este ejemplo de fichero `.ly`, es posible dar un acercamiento a la notación que utiliza Lilypond para crear las partituras.

B.1. Version

Lilypond es un software al que se le siguen aplicando actualizaciones, por tanto es recomendable indicar la versión a utilizar al inicio de cada fichero de este tipo de notación. Indicándose con `\version` seguido del número de la versión entre comillas. (Línea 1)

B.2. Formato

Con el comando `\paper` es posible denotar el formato que tendrá la partitura una vez se convierta a PDF. Incluyendo en este caso el espaciado entre líneas mínimo, máximo, el relleno entre notas etc. (Líneas 29-33)

B.3. Pentagrama

Para generar una partitura se usa el comando `\score`, con el que se da inicio a la posibilidad de editar ritmos, compases, claves, pentagramas, entre otras funciones. Con el comando `\new Staff` es posible indicar una nueva entrada en un pentagrama. (Líneas 36-53)

Claves

Existen muchos tipos claves que puedan representarse en Lilypond, pero las más usadas y aquí referenciadas son las siguientes:

- **Clave de Sol**: representada con el símbolo 𝄞, y como `\clef treble` en Lilypond. Se utilizará dentro de la notación de este proyecto al ser la clave donde se representa la melodía. (Línea 41)
- **Clave de Fa**: se la representa con el símbolo 𝄢, y como `\clef bass` en Lilypond. Pudiendo aparecer en la 3ª o 4ª línea del pentagrama, se utilizará esta última utilizará dentro de la notación de este proyecto al ser la clave donde se representa la melodía. (Línea 48)
- **Clave de Do**: representada con el símbolo 𝄡, y `\clef C` en Lilypond. Pudiendo aparecer en la 2ª, 3ª o 4ª línea del pentagrama, no se utilizará ya que no es útil a la hora de querer representar la escritura para acordeón.

Otras opciones

El resto de opciones utilizadas simbolizan lo siguiente:

- `\remove 'Time_signature_engraver'`: se eliminan la indicación de marcador de compás, ya que en la traducción lenguaje “zenbakizkopartitura no hay referencias a medidas ni ritmos. (Líneas 39, 46)

- `\set fingeringOrientations = #'(down/up)`: orientación de la digitación y demás marcas que aparecerán en la partitura. (Líneas 40, 47)
- `\penta_sol/penta_fa`: son las variables donde se guardan la melodía y el bajo respectivamente. Se detallarán más adelante. (Líneas 42, 49)
- `\bar '|.'`: El final de la partitura, en ambas líneas sonoras, tanto melodía como bajo. (Líneas 43, 50)

B.4. Notas

Como se menciona en la Sección 5.2, el registro `\penta_sol` guardará la información correspondiente a la melodía, y `\penta_fa` a las notas y acordes del bajo. Ambos se inicializan al principio para no poder dejar margen a generar errores. (Líneas 2-26)

Nombre

Para definir una nota, se necesitan entre otras cosas su nombre. Los posibles casos son los siguientes:

Lilypond	`c`	`d`	`e`	`f`	`g`	`a`	`b`	`r`
Partitura	do	re	mi	fa	sol	la	si	silencio

Tabla B.1: Notas musicales en Lilypond

Mencionar también las alteraciones, las cuales se escriben después del nombre de la nota:

- Si es un sostenido ♯: `is`
- Si es doble sostenido ♯♯: `isis`
- Si es un bemol ♭: `es`
- Si es doble bemol ♭♭: `eses`
- Si es becuadro ♮: se deja vacío

Altura

La altura de las notas puede depender de dos situaciones:

- El modo de entrada de octava **relativa** específica cada octava en relación a la nota anterior, si se cambia la octava de una nota afectará a todas las notas siguientes. Este modo se debe introducir de forma explícita usando la instrucción `\relative`.
- Las notas cuyos nombres van desde `c` hasta `b` se imprimen en la octava inferior al Do central. En la escritura de octava **absoluta** se utiliza la comilla (`'`) por cada octava que se quiera subir, y una coma (`,`) por cada una que se quiera bajar

Como es posible ver en el código anterior, se ha decidido usar la escritura de octava **absoluta** al poder generar un resultado tras la compilación que no tenga riesgos a la ambigüedad del modo relativo.

Medida

Para definir la medida regular se utilizan números, siendo el 1 la redonda, y las siguientes mitades, el doble del número anterior. Véase la Tabla B.2 para más claridad.

Lilypond	1	2	4	8	16	32	64
Partitura	𝅝	𝅗𝅥	♩	♪	𝅘𝅥𝅯	𝅘𝅥𝅰	𝅘𝅥𝅱

Tabla B.2: Medidas en Lilypond

Cuando se quieren crear medidas de valoración especial se usa el comando `\tuplet` para indicar cuantos sonidos se deben indicar en cuantos tiempos. (Línea 9)

Digitación

Al querer representar la digitación y notación especifica de la trikitixa junto a la partitura final, se ha decidido incluir las siguientes notaciones para indicarlo:

- `\finger\markup{\override #'(thickness . 1)}`: notación para añadir un círculo junto a la nota de la melodía correspondiente. Indicando que viene de una nota de acordeón con el fuelle cerrando. (Línea 24)
- `\finger\markup{\override #'(thickness . 5)}`: igual que el anterior, esta vez indicando que viene de una nota de acordeón con el fuelle abriendo. (Línea 5)
- `\circle{1}`: es el valor que se le da al círculo anteriormente definido. Corresponde al número de la nota en escritura de acordeón.
- `' '`: la digitación se realiza con el dedo índice (vacío).
- `\draw-circle #0.25 #0 ##f`: la digitación correspondiente a la nota de la melodía, en este caso siendo con el dedo corazón (circulo negro). (Línea 5)
- `\draw-circle #0.25 #0 ##t`: es la digitación correspondiente a el dedo anular (circulo blanco). (Línea 14)
- `\abs-fontsize #3 x`: digitación correspondiente al dedo meñique (cruz). (Línea 18)

Cifrado americano

El cifrado americano es un tipo de notación que suele aparecer en todo tipo de partituras y aquí se ha querido añadir como un extra para la ayuda del intérprete. Se colocará el correspondiente cifrado al final de cada entrada del bajo, `_\markup{ \abs-fontsize #10 G }` . Donde 'G' es el cifrado correspondiente al acorde de Sol Mayor. (Línea 5)

B.5. Signos de repetición

Existen dos tipos de repeticiones aquí implementadas:

- Para la 'Coda' se usa la expresión `\once \override Score.RehearsalMark.font-size = #4 \mark \markup { \musicglyph # scripts.coda }`. (Línea 22, 23)
- Para el 'Segno' se usa la expresión `\once \override Score.RehearsalMark.font-size = #3 \mark \markup { \musicglyph # scripts.segno }`.
- Entre las repeticiones de compás se tienen:
    - El fin de repetición `\bar ':|.'` (Línea 10, 11)
    - El inicio de repetición `\bar '.|:'`
    - La doble repetición `\bar ':..:'`
    - El fin de sección `\bar '||'`

B.6. Resultado

El resultado en PDF de el código Lilypond es el de la Figura B.1.

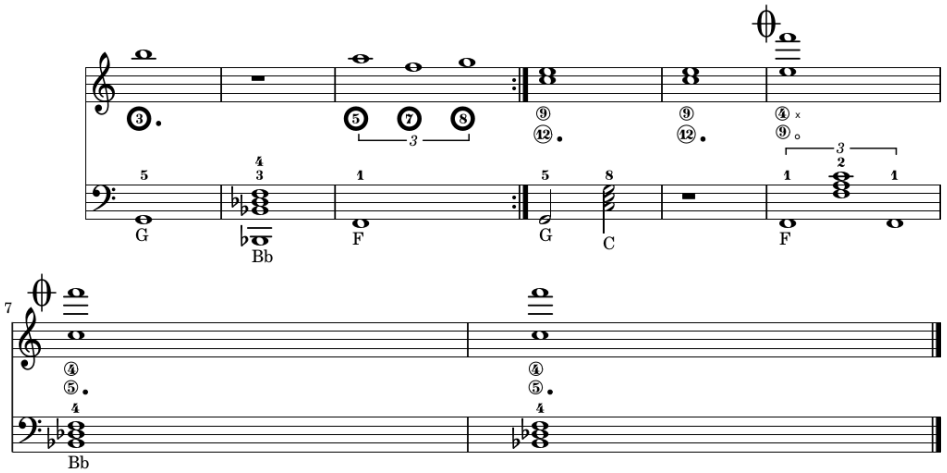


Figura B.1: Resultado en partitura del código Lilypond

APÉNDICE C

Gramática del lenguaje intermedio

En este apéndice se mostrará la gramática utilizada para el compilador de lenguaje intermedio. Es el contenido del fichero Bison (`.y`), dejando únicamente la estructura de símbolos terminales y no-terminales.

```
stmts :
       | stmt PUNTOCOMA stmts
       | INICIO_TRESILLO stmts
       | FIN_TRESILLO stmts
       | INICIO_REPETICION stmts
       | FIN_REPETICION stmts
       | DOBLE_REPETICION stmts
       | FIN_SECCION stmts
       | SEGNO stmts
       | CODA stmts

stmt :
       | nota_sol
       | nota_sol GUION nota_fa

nota_sol :
       | NUMERO fuelle parentesis_abrir sol_num_digits parentesis_cerrar
       | fuelle parentesis_abrir sol_num_digits parentesis_cerrar
```

```
19
20 nota_fa :
21        parentesis_abrir fa_medida parentesis_cerrar
22
23 parentesis_abrir :
24        PARENTESIS_ABRIR
25
26
27 parentesis_cerrar :
28        PARENTESIS_CERRAR
29
30 fuelle :
31        | FUELLE_ABRIR
32        | FUELLE_CERRAR
33
34 sol_num_digits :
35        sol_numero_digitacion sol_num_digits2
36
37 sol_num_digits2 :
38        | COMA sol_num_digits
39
40 sol_numero_digitacion :
41        sol_numero digitacion
42
43 sol_numero :
44        NUMERO
45
46 digitacion :
47        | DIGIT_VACIO
48        | DIGIT_BLANCO
49        | DIGIT_NEGRO
50        | DIGIT_X
51
52 fa_medida :
53        fa_medida_numero fa_medida2
54
55 fa_medida2 :
```

```
56        | COMA
57        | GUION fa_medida
58
59  fa_medida_numero :
60        | NUMERO
61        | NUMERO PUNTO_DOBLE NUMERO
```