

Sistema de conversión de Voz a MIDI

Ismael Mosquera Rivera

Proyecto Fin de Carrera

Director/a: Xavier Amatriain Rubio

Junio 2004

Ingeniería Técnica en Informática de Sistemas

Universitat Pompeu Fabra

A mis padres y hermanos.

Agradecimientos a Xavier Amatriain, Alex Loscos, Maarten de Boer,
y a todos los profesores y compañeros que me han ayudado durante mi carrera.

Resumen

En esta memoria se describe el análisis, diseño e implementación de un sistema de conversión de voz a MIDI. El interés, tanto a nivel académico como por las aplicaciones prácticas ofrecidas por este tipo de aplicaciones, ha animado al desarrollo de sistemas QBH (Query-by-Humming) durante la última década. Los sistemas QBH son utilizados para la recuperación de información, sobre piezas musicales contenidas en una base de datos, a partir de una señal de voz cantada por parte del usuario. La señal de entrada puede o no incorporar lyrics (letra de la canción); cuando no existen lyrics el usuario tararea ta/da...

El sistema implementado en este proyecto, realiza la transcripción de una señal de voz cantada mediante tarareo (sin lyrics), y obtiene una representación MIDI de la melodía contenida en dicha señal. También se han incorporado mecanismos de reproducción, visualización, edición y almacenamiento de resultados; todo ello ofrecido al usuario a través de una interfaz gráfica amigable. Para la implementación de las funcionalidades ofrecidas por el núcleo del sistema, E/S de audio, análisis y salida MIDI, se ha empleado el framework CLAM (C++ Library for Audio and Music), desarrollado por el MTG (Music Technology Group) en la Universitat Pompeu Fabra; y en el caso de la interfaz gráfica de usuario se ha hecho uso del toolkit gráfico Qt ofrecido por trolltech.

En los capítulos que forman este documento se discute sobre asuntos tales como: las técnicas de análisis espectral utilizadas para la obtención de la melodía contenida en la señal de audio de entrada, el funcionamiento del protocolo MIDI, las tecnologías orientadas a objetos empleadas en el análisis y diseño del sistema, las herramientas utilizadas, el lenguaje de programación empleado, y demás aspectos relacionados con el desarrollo de la aplicación. En el capítulo dedicado a las conclusiones se ha incluido un pequeño estudio de usabilidad, realizado a partir de la opinión de varios usuarios, acerca del funcionamiento y la calidad en los resultados ofrecidos por el sistema.

Índice

1. Introducción.....	9
2. Conceptos fundamentales.....	11
2.1 Sobre la producción y percepción de la voz.....	11
2.1.1 El aparato fonador.....	12
2.1.2 El aparato auditivo.....	13
2.2 Análisis espectral.....	15
2.2.1 La transformada de Fourier.....	16
2.2.2 La DFT y la FFT.....	16
2.2.3 La STFT.....	18
2.2.4 ¿Por qué usar la STFT?.....	19
2.2.5 Parámetros de la STFT.....	19
2.2.6 Detección de frecuencia fundamental y cálculo de la energía.....	20
2.2.7 Segmentación.....	21
2.3 MIDI.....	21
2.3.1 Canales.....	22
2.3.2 Mensajes.....	22
2.3.3 Control.....	23
3. Requisitos de la aplicación.....	25
3.1 Requisitos funcionales.....	25
3.2 Requisitos no funcionales.....	28
3.3 Valoración de la viabilidad del sistema.....	28
4. Herramientas empleadas.....	31
4.1 C++.....	31
4.2 Compiladores.....	35
4.3 CLAM: C++ Library for Audio and Music.....	36
4.4 Qt: el toolkit gráfico de Trolltech.....	39
5. Diseño e implementación.....	43
5.1 Introducción.....	43
5.2 Arquitectura interna.....	45
5.2.1 E/S de audio.....	46
5.2.2 El analizador y el segmentador.....	50
5.2.3 Soporte MIDI: conversión y salida.....	53
5.3 Visualización.....	57
5.4 XML.....	63
5.5 El sistema de ayuda.....	64
5.6 Uniendo todas las piezas.....	65
6. Conclusiones.....	71
6.1 Objetivos conseguidos y estado actual.....	71
6.2 Estudio de usabilidad.....	71
6.3 Comparativa con aplicaciones similares.....	73
6.4 Desarrollo futuro.....	74
Anexo---.....	75
A.1 Planificación.....	75
A.2 Escenarios de casos de uso.....	78
A.3 Cuestionario de usabilidad.....	89
A.4 Instrucciones para la compilación del programa.....	96
Bibliografía y referencias web.....	97

1. Introducción

En la primera etapa de mi adolescencia, tuve la oportunidad de adquirir cierto conocimiento sobre como interpretar una partitura musical, en mi caso, el instrumento elegido fue la trompeta. Disfruté mucho practicando con ese instrumento, pero lamentablemente acabé por dejarlo; todavía conservo mi apreciada trompeta descansando en el interior de su estuche, esperando a tener la oportunidad de volver a deleitarme con su uso. Sin embargo, no fue hasta que cursé mis estudios de ingeniería informática en la Universitat Pompeu Fabra, cuando empecé a descubrir la forma de descomponer un sonido para poder ver lo que encierra en su interior.

El campo de estudio de las técnicas de análisis y tratamiento de señales es muy amplio y está en constante evolución. El dominio en el que se encuentra el contexto de este proyecto es relativamente joven, dado que la implementación eficaz de los algoritmos requeridos necesita gran potencia y rapidez de cálculo, así como elevados recursos de memoria, lo cual está siendo posible gracias al avance tecnológico, que crece de forma exponencial.

Existe software en el mercado que realiza la conversión de voz a MIDI (Musical Instrument Digital Interface), incluso en tiempo real; asimismo, también existen micrófonos MIDI y hardware que también implementan esta funcionalidad, aunque en este terreno se continúa investigando para lograr obtener resultados más precisos, sobre todo en el problema de la conversión en tiempo real, donde existe un inevitable retardo en la respuesta del sistema que en ocasiones puede resultar crítico, además, en los sistemas en tiempo real se pasan por alto detalles importantes como las pequeñas micro variaciones que se producen en la señal sonora para poder ganar rapidez de respuesta, hecho que desemboca en una menor fidelidad en los resultados obtenidos.

La finalidad de este proyecto es la implementación de un sistema cuyo núcleo principal lleve a cabo esta conversión de voz a MIDI fuera de tiempo, es decir, no en tiempo real sino a partir de un sonido previamente almacenado, o registrado a través de un micrófono haciendo uso del programa, el cual también debe incorporar esta funcionalidad. Además, el sistema también ofrecerá al usuario la posibilidad de visualizar de forma gráfica los resultados obtenidos en el análisis del sonido, así como la reproducción del audio original y del resultado obtenido tras la conversión mediante el protocolo MIDI.

Entonces, durante el proyecto se debe buscar solución a diversos problemas de distinta índole, como son: realizar un análisis de la señal a través del cual podamos obtener unos parámetros de fidelidad aceptable respecto del sonido analizado, para que después de llevar a cabo su conversión a mensajes MIDI el resultado se asemeje lo máximo posible a la información realmente contenida en la entrada de audio original; la visualización gráfica de resultados; la captura de la señal a través de un micrófono; la reproducción de audio y MIDI; el almacenamiento y carga de archivos ... La solución a la mayor parte de estos problemas nos la ofrecerá el framework CLAM: C++ Library for Audio and Music, del cual se hablará en la sección dedicada a las herramientas utilizadas en el proyecto. Ahora, vamos a describir un poco la estructura de esta memoria.

La memoria está dividida en varias secciones la primera de las cuales es esta introducción. En la segunda sección se muestran los conceptos básicos necesarios para tener un punto de partida desde el cual afrontar el problema en el que se centra este proyecto. En la tercera sección se realiza un análisis de los requisitos de la aplicación donde también se decide su viabilidad. La sección cuarta trata sobre las herramientas empleadas para el desarrollo del producto final, tales como frameworks, toolkits gráficos, etc. La quinta sección está dedicada al diseño e implementación de la aplicación. En la sexta sección se comentan las conclusiones extraídas y las posibles mejoras que se pueden hacer sobre el trabajo realizado. A continuación se ha colocado un anexo en el cual se incluye la planificación del proyecto y diverso material; también se indica como construir la aplicación a partir del código fuente. Finalmente, se incluye un apartado dedicado a la bibliografía y referencias web que se han consultado durante el desarrollo del proyecto.

2. Conceptos fundamentales

En este capítulo se presentan de forma somera e informal los temas básicos que se deben tener en cuenta en el proyecto, ya que forman parte del dominio en el que está situado.

No se comentan aquí otras áreas que forman parte del contexto de la aplicación que nos ocupa, como pueden ser las herramientas que se emplearán para el desarrollo del programa, ya que esto será descrito en un capítulo posterior. Ahora lo que se introduce es una explicación sobre los diferentes campos en que se sitúa el ámbito del proyecto, que pueda ofrecer una visión global del problema al cual nos enfrentamos, y que nos indique las líneas a seguir para poder implementar una solución adecuada y eficaz.

En primer lugar, se describe de forma breve el funcionamiento de la producción de la voz y la percepción sonora desde una perspectiva global. A continuación, se presentan los conceptos relacionados con el análisis espectral que entran en el ámbito del proyecto, y para finalizar, se expone una visión general del funcionamiento del protocolo MIDI.

2.1 Sobre la producción y percepción de la voz

El sonido se define como la sensación producida en el órgano del oído por el movimiento vibratorio de los cuerpos, transmitido por un medio elástico como el aire. Entonces podemos deducir que el origen del sonido es la vibración de los cuerpos. Se dice que la vibración es periódica cuando se repite a intervalos regulares.

La representación matemática que describe el movimiento vibratorio que se produce cuando golpeamos un diapasón puede expresarse mediante la función coseno:

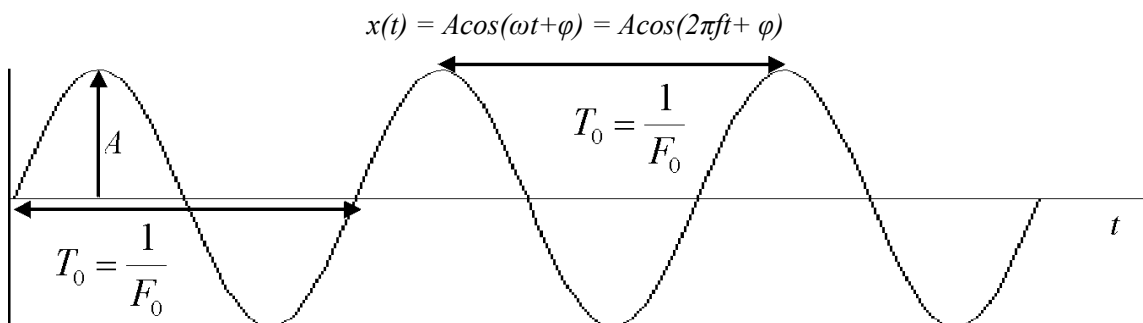


Fig 1. Movimiento armónico simple.

A: amplitud, T: periodo, F: frecuencia

Para generar esta función mediante un ordenador debemos pasar la variable continua t a la forma discreta nT , con $n = 0, 1, \dots$ y donde T representa el periodo de muestreo o intervalo entre dos muestras, siendo la frecuencia de muestreo $F_s = 1/T$. La diferencia entre la forma continua y la discreta es que la segunda no está definida en todo el eje temporal, sino que sólo se define en instantes concretos equiespaciados en el tiempo.

Cualquier sonido complejo puede ser formado como la suma de armónicos simples de distinta frecuencia, amplitud y fase; que corresponden a los parciales de la señal sonora. Así pues, para analizar una señal de audio debemos descomponerla en el conjunto de armónicos simples que la componen, ya que sus intensidades y frecuencias nos proporcionan información sobre las características del sonido original.

La mayoría de sonidos producidos por la voz son cuasiperiódicos, por tanto, podemos aplicar este concepto para su análisis. Por otra parte, el sonido aperiódico o ruido es una onda que no presenta ninguna periodicidad y al descomponerla se observa que la energía está dispersa entre sus armónicos y carece de estabilidad.

2.1.1 El aparato fonador

El órgano productor de la voz se compone de tres partes diferentes: las cavidades infraglólicas, la cavidad glótica y las cavidades supraglólicas; cada una de ellas realiza una función determinada pero todas ellas son necesarias para la producción del habla.

La misión de las cavidades infraglólicas es proporcionar la corriente de aire necesaria para producir el sonido, y se componen de diafragma, pulmones, bronquios y tráquea. El diafragma es un músculo que se encuentra situado por debajo de los pulmones y tiene forma de cúpula, cuyo trabajo es controlar, junto con los músculos pectorales el hinchado y vaciado de la cavidad pulmonar produciendo así la respiración. Cuando el diafragma se contrae, se ensancha la cavidad torácica y se produce la inspiración de aire, y cuando se relaja se reduce la cavidad torácica y se produce la expiración del aire contenido en los pulmones. Los bronquios y la tráquea se encargan de conducir el aire desde los pulmones hacia la laringe, y su cometido es actuar como canales de transmisión del flujo aéreo. El funcionamiento del aparato respiratorio nos sugiere la idea de un compresor.

La cavidad glótica está formada por la laringe, y tiene como característica especial la presencia de las cuerdas vocales, las cuales son las responsables de producir la vibración básica para la generación del sonido. Las cuerdas vocales son en realidad un par de pliegues cubiertos por una membrana mucosa; tienen una longitud aproximada de 3 mm en recién nacidos, y de 9 a 13 mm y 15 a 23 mm en mujeres y hombres adultos respectivamente. La importancia de la longitud de las cuerdas vocales en la producción del tono de voz ha sido científicamente analizada recientemente por Sawashima (1983), quien mostró que a mayor longitud, la voz tiene un rango de tonalidades más grave, y a menor longitud, dicho rango está formado por tonalidades más agudas.

Cuando la corriente de aire que proviene de las cavidades infraglólicas pasa por la glotis (espacio que queda entre las cuerdas vocales) las hace vibrar, y el tono resultante puede variar en frecuencia e intensidad dependiendo de la presión del aire y de la longitud y masa de las cuerdas vocales; por ejemplo, si un cantante está entonando un A4 (nota La de la cuarta octava), las cuerdas vocales se abren y cierran 440 veces por segundo, es decir, oscilan con una frecuencia de 440 Hz. Si queremos hacer una buena descripción de las características de la voz debemos definirla en tres dimensiones: frecuencia fundamental, amplitud y espectro; o en términos de acústica: tono, intensidad y timbre. Según esta exposición, en términos de ingeniería podríamos pensar en esta parte del aparato fonador como si se tratase de un oscilador.

Las cavidades supraglólicas son: faríngea, nasal, bucal y labial. A continuación de la laringe se encuentra la faringe, de donde arranca la raíz de la lengua, y aparece el primer obstáculo móvil: la úvula, que es el apéndice situado al final del velo del paladar o paladar blando. Si la corriente de aire sale exclusivamente por la boca se producen sonidos orales, pero si el velo del paladar está caído, parte del aire será expulsado por la cavidad nasal. La cavidad nasal carece de elementos móviles, por lo cual tiene una función pasiva en la producción del habla. La lengua es el órgano más móvil de la boca registrando, una elevada actividad durante la producción del habla y está dividida en tres partes: raíz, dorso y ápice. Recientemente se ha demostrado que dependiendo del perfil que adopta se produce un resonado acústico, por lo que el timbre del sonido será diferente según la forma sea convexa, cóncava o plana, o esté situada en la parte anterior, central o posterior. Los dientes son elementos pasivos en la actividad del habla por estar colocados de forma fija en los maxilares; los inferiores tienen movilidad por estar sujetos en la mandíbula inferior, siendo ésta activa en la articulación. El paladar consta de dos zonas: el paladar duro, situado sobre el hueso palatino, y el paladar blando, mencionado anteriormente, el cual termina en la úvula. Por último tenemos los labios, los cuales poseen bastante movilidad y permiten la modificación de los sonidos.

Por tanto, podemos ver que para la producción del habla son necesarias una fuente de energía que expulse aire a presión (aparato respiratorio), un órgano vibratorio (las cuerdas vocales), y una caja de resonancia con sistema de articulación.

A partir de esta información general sobre el funcionamiento del aparato fonador, y una vez

marcada la separación en diferentes bloques de las distintas partes que toman parte en la producción del habla podemos representar el concepto mediante el diagrama de la figura 2.

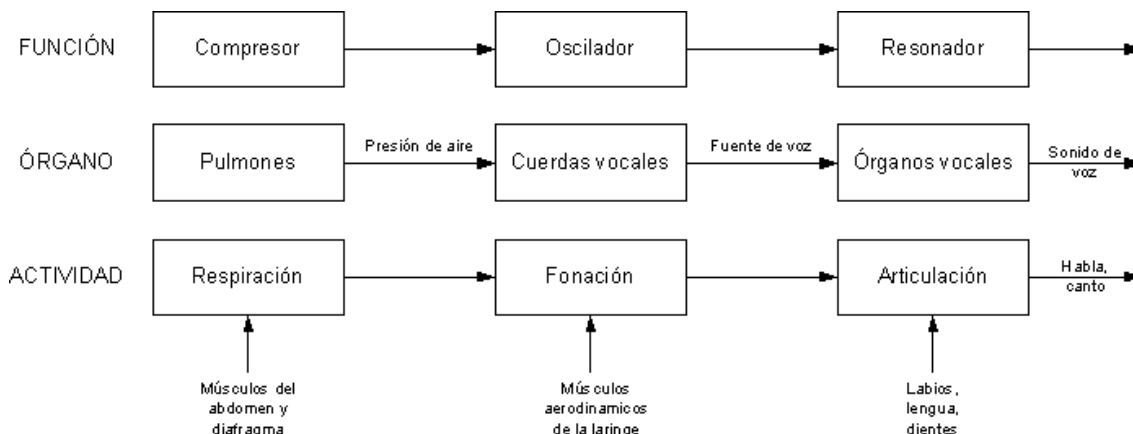


Fig 2. Representación del aparato fonador mediante diagrama de bloques.

El proceso se inicia con la expiración del aire, que cuando pasa a través de las cuerdas vocales las hace vibrar a una determinada frecuencia dependiendo de su tensión; esta frecuencia se denomina frecuencia fundamental que es la que determina el tono, si la frecuencia es baja el tono será grave, y si la frecuencia es alta el tono será agudo. Según como se encuentren formados los órganos se formará una caja de resonancia que potenciará un conjunto de frecuencias y atenuará el resto, por tanto esta caja actúa como un filtro de frecuencias.

2.1.2 El aparato auditivo

El rango de frecuencias que puede percibir el ser humano se sitúa entre los 20 Hz y los 20000 Hz. En una onda, la intensidad es relativa a la energía (o variación de presión de aire). En general, cuando hay un incremento de intensidad también hay un incremento en la intensidad sonora (atributo perceptual). La percepción de la intensidad sonora tiene un comportamiento exponencial, por lo cual, para medirla es preferible usar una escala logarítmica. La intensidad sonora se mide en términos del nivel de presión sonora (SPL: sound pressure level), que se define como

$$SPL = 20\log_{10}(P/P_0)$$

donde la presión de referencia P_0 , corresponde a la presión de un tono sinusoidal de 1000 Hz en el umbral de audición, y es igual a $2 \cdot 10^{-5}$ newtons/metro². El nivel de presión sonora se expresa en dB, que es la unidad de medida para la intensidad sonora.

El oído es más sensible a algunas regiones frecuenciales que a otras. La región más sensible está entre los 2700 y 3200 Hz, es por eso que una onda sinusoidal a 3000 Hz con una determinada intensidad se percibe con una sonoridad mucho mayor que otra a 200 o 8000 Hz de igual intensidad. Una buena referencia acerca de esta característica nos la ofrecen las curvas de Fletcher-Munson, las cuales forman contornos de sonoridad constante llamados contornos phon, como funciones de la frecuencia. Los phons se corresponden con los decibelios a 1000 Hz. Un tono a 1000 Hz con una intensidad sonora de 40 dB SPL tiene un nivel de sonoridad de 40 phons; si queremos producir un tono a 100 Hz con el mismo nivel de sonoridad y miramos la gráfica, podemos descubrir que necesitamos aproximadamente 62 dB SPL.

Las siguientes figuras muestran una representación del área de audición humana como función de la frecuencia y de la intensidad, y la gráfica de las curvas de Fletcher-Munson respectivamente.

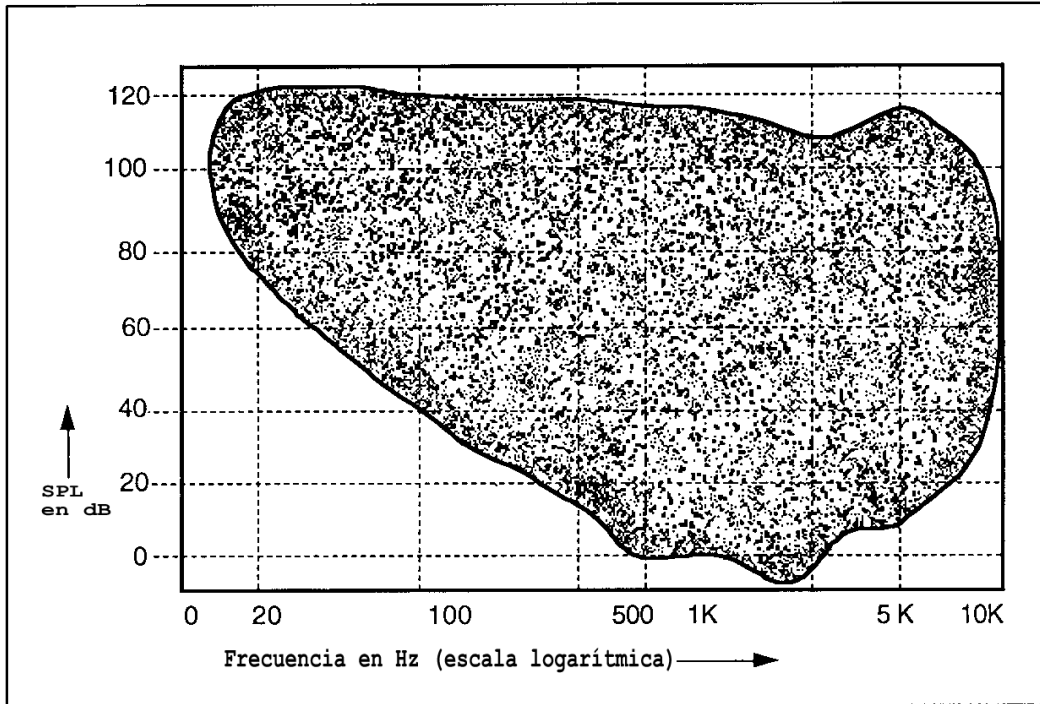


Fig 3. Área de audición humana.

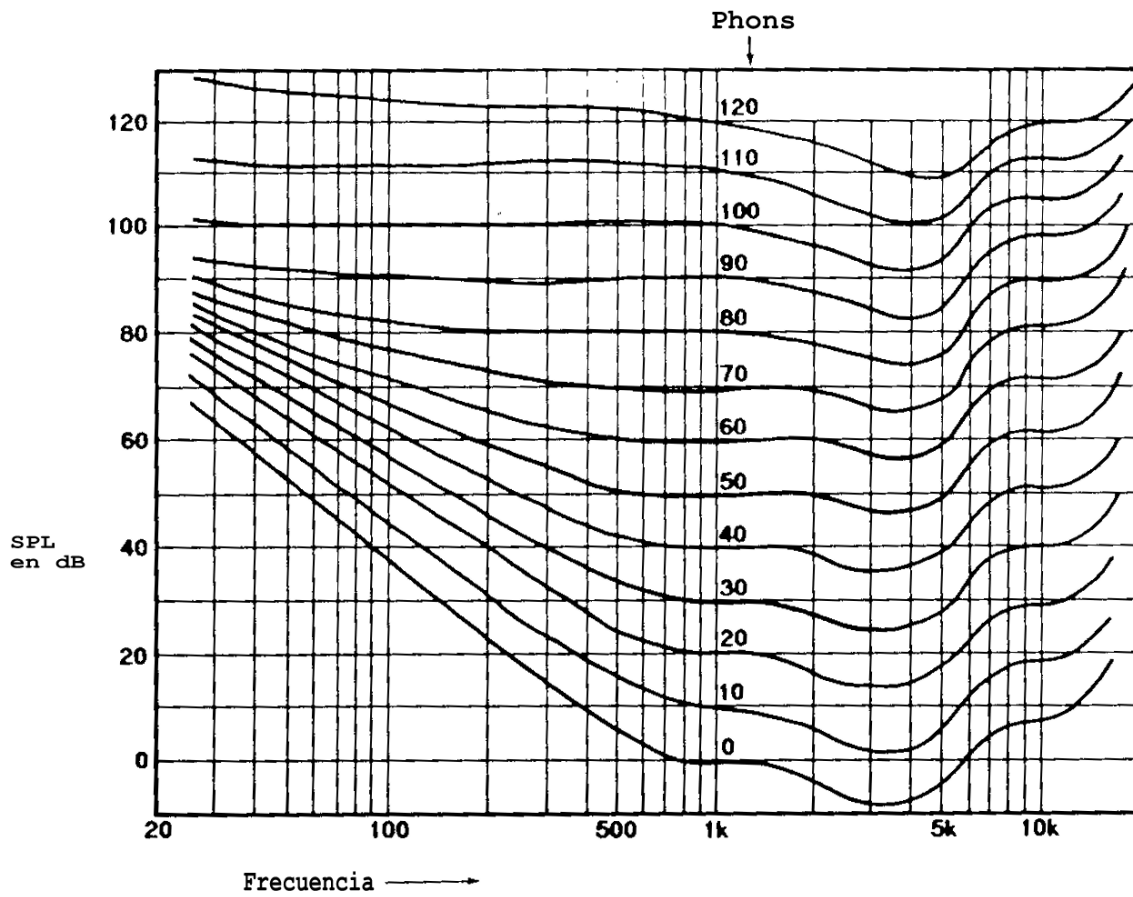


Fig 4. Curvas de Fletcher-Munson.

La función del oído es capturar las ondas acústicas y transformarlas en impulsos nerviosos que el cerebro puede interpretar. Está formado por tres partes: el oído externo, el oído medio y el oído interno.

El oído externo está formado por el pabellón auditivo (oreja), y por el conducto auditivo externo. El pabellón auditivo recoge las ondas sonoras y facilita su paso hacia el interior, aplicando una amplificación sobre dichas ondas. El conductor auditivo externo termina en el tímpano, que es una membrana que separa el oído externo del oído medio. El conductor auditivo conduce las ondas hacia el tímpano atenuando los sonidos más agudos que puedan causar daño.

El oído medio está separado del oído externo a través del tímpano, y dispone de una cadena ósea de tres huesecillos: martillo, yunque y estribo. En el oído medio también se encuentra la trompa de Eustaquio, que es un canal que comunica con la faringe. La misión del oído medio es transmitir al oído interno los sonidos que llegan desde el oído externo realizando una adaptación. Cuando la intensidad es pequeña, la cadena ósea se mueve en conjunto produciendo un aumento de la misma; cuando la intensidad es grande, se produce una disminución de la misma con el fin de evitar daños en el oído interno.

En el oído interno se encuentra el caracol, que es el órgano de audición, y cuya función es percibir las frecuencias de las vibraciones sonoras y convertirlas en impulsos nerviosos que transmite al cerebro para su interpretación.

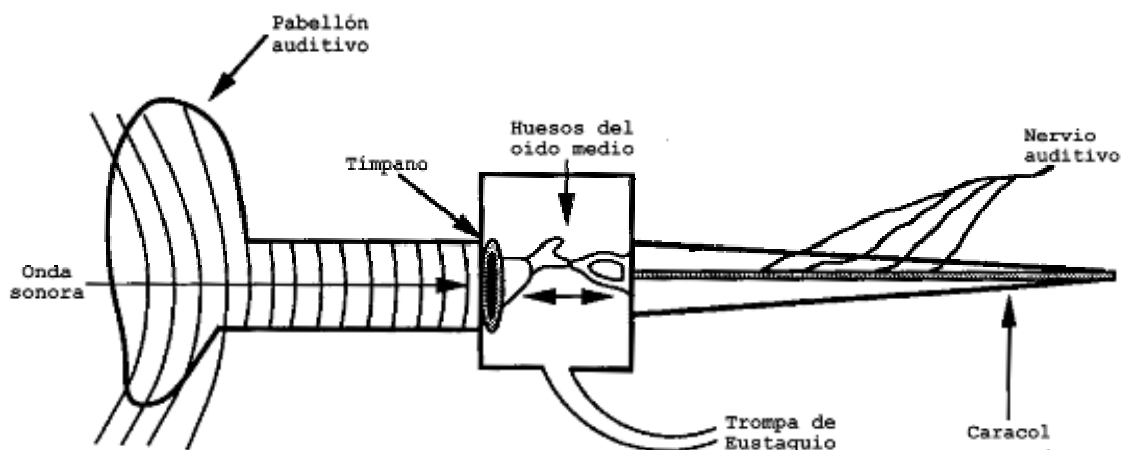


Fig 5. Representación del aparato auditivo.

La figura sobre estas líneas es una representación gráfica del aparato auditivo. En primer lugar, el sonido es amplificado, para posteriormente equilibrarse; el sonido se transforma en su representación a través del caracol, del cual sale el conjunto de señales nerviosas que llega al cerebro.

2.2 Análisis espectral

En el siglo XIX, científicos y músicos ya conocían que muchos sonidos musicales estaban caracterizados por vibraciones armónicas alrededor de un tono fundamental, pero no disponían de la tecnología necesaria para analizar esos armónicos de una forma sistemática. Isaac Newton empleó el término “espectro”, para describir las bandas de color que muestran las diferentes frecuencias cuando se descompone la luz blanca al pasar a través de un prisma de cristal.

Tal como una imagen puede describirse como una mezcla de colores (frecuencias en la parte visible del espectro electromagnético), un sonido puede ser descrito como una combinación de vibraciones acústicas elementales. Una forma de diseccionar un sonido es considerar la contribución de varios componentes, cada uno correspondiente a una cierta variación en la presión del aire. Medir la distribución entre estos componentes se denomina análisis espectral.

El análisis espectral revela las características de frecuencia y energía en tonos vocales e instrumentales, lo cual lo hace valioso en la detección de la tonalidad, factor imprescindible en el contexto de este proyecto. La gráfica del espectro de una señal sonora se representa en un plano bidimensional cuyos ejes son la amplitud y la frecuencia. De entre los métodos existentes empleados para la realización de un análisis espectral, el más ampliamente utilizado es el análisis de Fourier, el cual se comenta brevemente a continuación.

2.2.1 La transformada de Fourier

En 1822 el ingeniero francés Jean-Baptiste Joseph, Baron de Fourier publicó su tesis sobre teoría analítica del calor; en ella describía como complicadas vibraciones pueden ser analizadas como suma de varias señales simples; en particular, demostró que cualquier función periódica puede ser representada como una suma infinita de senos y cosenos.

La transformada de Fourier es una herramienta matemática que nos permite obtener el espectro de una señal; la formulación de la integral de Fourier es

$$X(f) = \int_{-\infty}^{\infty} x(t)e^{-j\omega t} dt$$

lo que indica que la FT (Fourier Transform) de una determinada frecuencia f es la integral de la multiplicación de la señal de entrada $x(t)$ por el tono puro $e^{-j\omega t}$. El espectro de frecuencias entre 0 Hz hasta infinito con su correspondiente imagen especular en el eje negativo de frecuencias es $X(f)$, que es la notación empleada para representar la FT.

Los sistemas digitales tratan con señales discretas, por esa razón es necesaria una representación en tiempo discreto de la FT (señal en tiempo continuo); dicha solución viene dada por la DTFT (Discrete Time Fourier Transform).

2.2.2 La DFT y la FFT

Una señal discreta es una señal analógica que ha sido muestreada. Después del muestreo, la señal pasa a través de un filtro anti-aliasing, con el fin de atenuar (eliminando al máximo) las frecuencias que están por encima de la mitad de la velocidad de muestreo (tasa de Nyquist), produciendo una nueva señal discreta. La notación para las señales en tiempo discreto se representa con el índice temporal entre corchetes: $x[n]$.

Así pues, si muestreamos una señal $x(t)$ con una frecuencia de muestreo F_s obtenemos la señal discreta $x[n]$. El periodo de muestreo T es igual a $1/F_s$ por lo que la señal resultante es $x(nT)$. La transformada de Fourier en tiempo discreto DTFT puede denotarse como $X(e^{j\theta})$, donde θ es la fase que toma valores positivos y negativos. La DTFT revela el espectro continuo de frecuencias de la señal $x[n]$, que sólo está definida para valores de la variable entera n , pero que está compuesta por un número infinito de muestras. Si tenemos en cuenta que una exponencial compleja es idéntica a otra que difiere de ella en frecuencia en múltiplos de 2π , entonces podemos representarla hasta el infinito en intervalos de 2π correspondientes a múltiplos de la velocidad de muestreo F_s . La DTFT puede ser representada como

$$X(e^{j\theta_0}) = X(e^{j(\theta_0 + 2\pi k)})$$

donde k es un múltiplo arbitrario. En la práctica sólo es necesario considerar los valores de la DTFT en un periodo de 2π , y se asume que la DTFT $X(e^{j\omega})$ entre $-\pi$ y π está libre de aliasing, pero $X(e^{j\omega})$ es un espectro infinito y una computadora puede calcular solamente cantidades discretas, por lo que necesitamos una técnica de análisis que trabaje con espectros discretos de señales discretas. Esta técnica es la transformada de Fourier discreta: DFT. Podemos construir la señal $\tilde{x}[n]$ replicando la señal $x[n]$ hasta el infinito, y la DFT de $\tilde{x}[n]$ se compone de muestras de la DTFT de $x[n]$. Así como muestrear $x(t)$ en el dominio temporal resulta en replicar en el dominio frecuencial; replicar $x[n]$ en el dominio temporal resulta en muestrear en el dominio frecuencial.

En las figuras siguientes se muestra este concepto de forma gráfica.

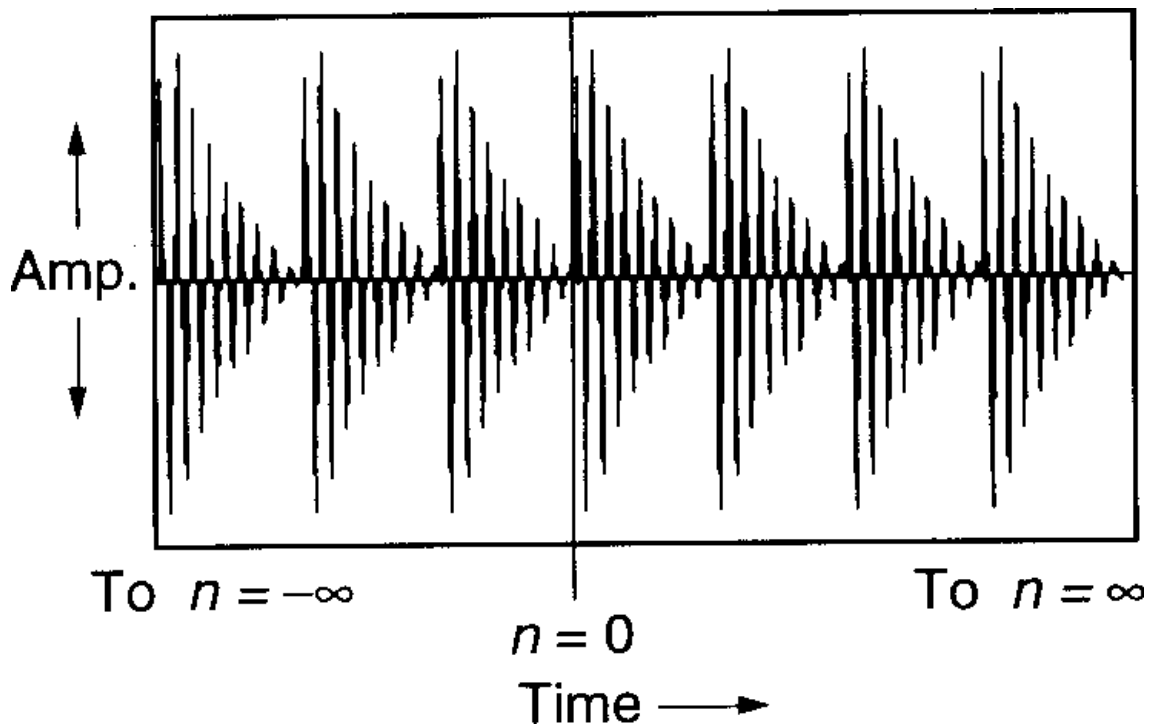


Fig 6. Señal muestreada $\tilde{x}[n]$

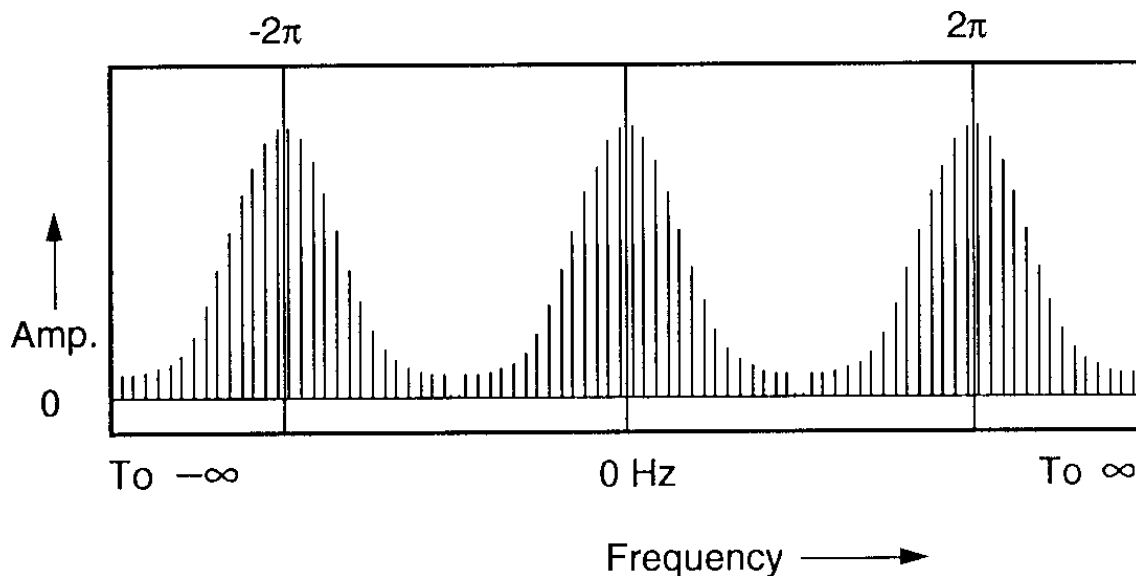


Fig 7. DFT de la señal muestreada $\tilde{x}[n]$

En la figura 6 se muestra que la señal $\tilde{x}[n]$ consiste en réplicas hasta el infinito de la señal $x[n]$. En la figura 7 vemos una gráfica de la DFT de $\tilde{x}[n]$, que resulta en un espectro discreto en frecuencia.

La FFT (Fast Fourier Transform) no es más que una implementación eficiente de la DFT para conseguir un rendimiento computacional aceptable. La DFT tiene un orden de N^2 multiplicaciones complejas, mientras que el orden de la FFT es de $N \log(N)$ operaciones, lo cual es una importante optimización en el rendimiento, si tenemos en cuenta además la gran cantidad de datos que intervienen en el análisis de audio.

2.2.3 La STFT

Con el fin de adaptar el cálculo de la FT a la práctica, se ha modelado un método conocido como short-time Fourier transform (STFT). Como preparación para el análisis espectral, la STFT impone aplicar una secuencia de ventanas temporales sobre la señal de entrada, es decir, fragmenta la señal de entrada en una serie de pequeños segmentos acotados en duración, mediante una función de enventanado. Una ventana no es más que un tipo de envolvente diseñada para el análisis espectral, y su duración se encuentra, habitualmente, en el rango de 1ms a 1s., produciéndose en ocasiones solapamiento entre los segmentos enventanados.

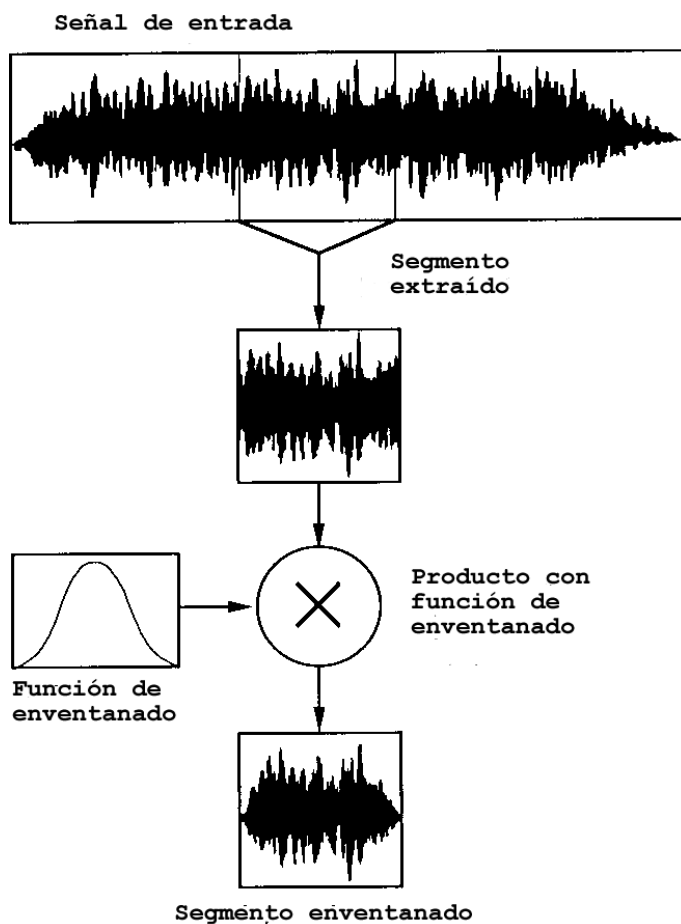


Fig 8. Enventanado de señal.

Después del enventanado, la STFT aplica la transformada de Fourier discreta a cada segmento, esto es, su implementación eficiente: la FFT, y cada segmento generado por la FFT se denomina frame. Cada frame contiene dos cosas: 1) el espectro de magnitud, que determina la amplitud de cada componente frecuencial en el espectro, y 2) el espectro de fase que indica el valor de la fase inicial de cada componente frecuencial. En resumen, la aplicación de la STFT a una señal muestreada resulta en una serie de frames que forman un espectro variante en el tiempo.

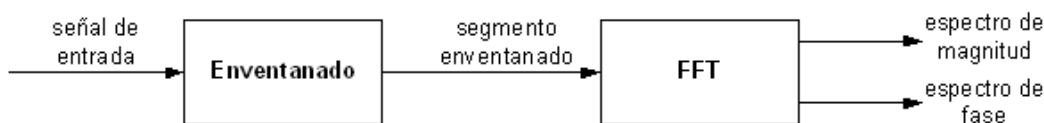


Fig 9. Esquema general de aplicación de la STFT.

2.2.4 ¿Por qué usar la STFT?

Existen varios motivos por los cuales es aconsejable (incluso necesario), el uso de la STFT para realizar el análisis espectral de una señal. Podríamos preguntarnos por qué tomarnos tantas molestias fragmentando la señal y aplicando ventanas si podemos aplicar la FFT a toda la señal para obtener una representación de su espectro, y que además podemos invertir el proceso mediante la IFFT para obtener la señal original sin pérdidas.

Los motivos son importantes. Por ejemplo, el análisis de una señal monoaural muestreada a una frecuencia de 44100 Hz y de 30 minutos de duración produce un espectro de alrededor de 79 millones de puntos. La inspección visual de tal espectro sería prácticamente imposible.

Otra razón importante es el uso de memoria. Si tuviéramos que analizar 30 minutos de audio muestreado con resolución de 16 bits, nos harían falta cerca de 79 millones de palabras de RAM para poder mantener los datos en memoria mientras se realiza el cálculo de la FFT, sin embargo, si segmentamos la señal en pequeños trozos es más sencillo calcular la FFT de cada fragmento uno tras otro.

Un motivo esencial por el que usar la STFT, es que de esta forma podemos obtener resultados de forma rápida. Enventanar la señal nos permite conocer resultados iniciales en pocos milisegundos a partir del comienzo del análisis, lo cual abre una puerta para aplicaciones de análisis espectral en tiempo real.

2.2.5 Parámetros de la STFT

Como podemos ver, aplicar la STFT a una señal requiere una preparación de ésta antes de calcular la FFT de cada segmento. Podemos considerar que los parámetros empleados en dicha preparación afectarán el resultado final del análisis de un modo u otro. Parámetros empleados en la aplicación de la STFT son:

- Tipo de ventana de análisis.
- Tamaño de ventana de análisis.
- Tamaño de avance de ventana (hop size).
- Zero-padding.

Existen distintos tipos de ventana que suelen ser usados en el cálculo de la STFT. Entre estos se encuentran: Hamming, Hanning, BlackmanHarris, Gaussian, etc. El tipo y tamaño de ventana utilizados afecta directamente sobre los resultados que se obtienen.

Las características principales del tipo de ventana son el ancho en bins (muestras espectrales) de su lóbulo principal, la diferencia de dB del punto más alto del lóbulo principal al siguiente lóbulo más cercano (sus vecinos izquierdo y derecho), y el decaimiento expresado en dBs por octava respecto del resto de lóbulos. En la siguiente figura se muestra una ventana Hamming, su espectro, y sus características principales

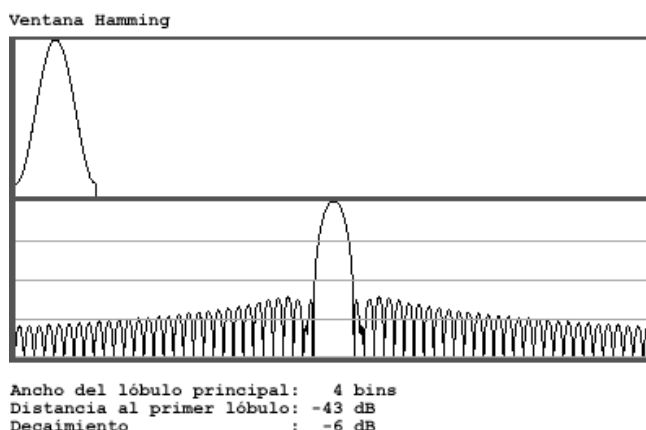


Fig 10. Ventana Hamming.

El tamaño determina el equilibrio existente entre la resolución temporal y frecuencial del espectro; a mayor tamaño mayor resolución frecuencial, y a menor tamaño mayor resolución temporal; ya que uno de los factores afecta directamente sobre el otro debe buscarse un tamaño adecuado según el tipo de resultado que desee obtenerse a partir del análisis. Para ello puede aplicarse la relación siguiente

$$M \geq B_s \frac{f_s}{\Delta} = B_s \frac{f_s}{|f_{k+1} - f_k|}$$

donde M es el tamaño de ventana, B_s es el ancho del lóbulo principal (en bins), y f_s es la velocidad de muestreo. Si f_k y f_{k+1} son armónicos de la frecuencia fundamental f_0 , entonces $f_0 = f_{k+1} - f_k = \Delta$.

En el proceso de enventanado suele aplicarse solapamiento entre las ventanas, con el fin de obtener una mejor calidad en los resultados mediante una ponderación equitativa, entonces, la representación matemática de la STFT puede expresarse mediante la siguiente fórmula

$$X_l(k) = \sum_{n=0}^{N-1} w(n)x(n+lH)e^{-jwkn}$$

w : ventana real

l : número de frame

H : avance temporal (hop-size)

El factor de zero-padding se utiliza con el fin de aplicar interpolación de frecuencias en el espectro; esto no produce variaciones en el resultado y ayuda en el proceso de detección de picos espectrales (spectral peaks). El zero-padding consiste en añadir ceros a la señal antes de aplicar la FFT. El tamaño del segmento de señal aumenta en potencias de 2 según el factor de zero-padding aplicado. Por ejemplo, si tenemos un segmento de 256 muestras:

factor de zero-padding = 0 → tamaño 256 (no hay variación).

factor de zero-padding = 1 → tamaño 512 (aumenta a la siguiente potencia de 2), se añaden ceros para completar el tamaño.

factor de zero-padding = 2 → tamaño 1024 (dos potencias de 2 más arriba; se añaden ceros).

...

2.2.6 Detección de frecuencia fundamental y cálculo de la energía

La STFT recibe como entrada una señal, y da como salida una secuencia de frames conteniendo cada una el espectro de la ventana de audio correspondiente al frame. Con el fin de obtener las características que nos interesan, para llevar a cabo la segmentación de la entrada, en las notas que formarán la melodía, que posteriormente convertiremos a mensajes MIDI, debemos hacer un análisis del espectro de cada frame para extraer dicha información; en este caso nos interesan la frecuencia fundamental y la energía.

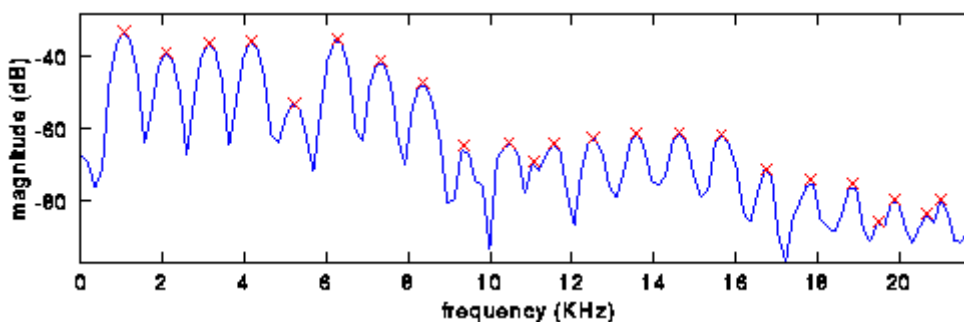


Fig 11. Picos en espectro de magnitud.

En primer lugar se debe realizar una detección de picos espectrales, los cuales corresponden a máximos locales en el espectro de magnitud, bajo restricciones en el rango de frecuencia y umbral de magnitud. Ya se ha comentado anteriormente, que aplicar un factor de zero-padding incrementa la resolución en el espectro devuelto por la FFT, favoreciendo así la detección de picos. Una vez realizada la detección de picos espectrales, se puede hacer una estimación de la tonalidad contenida en el frame, es decir, su frecuencia fundamental (pitch). Una forma de hacer esta estimación es definir la frecuencia fundamental como el común divisor de la serie armónica que mejor explica los picos espectrales.

El framework CLAM ofrece las herramientas necesarias para realizar tanto la detección de picos espectrales como la estimación de la frecuencia fundamental a partir de éstos. En el capítulo dedicado a las herramientas utilizadas en el desarrollo, se ofrece una visión general de este entorno de trabajo, enfocada a las necesidades de este proyecto.

La energía de cada frame la calculamos a partir del espectro mediante la relación de Parseval

$$E_x = \int_{-\infty}^{\infty} |x(t)|^2 dt = \int_{-\infty}^{\infty} |X(w)|^2 dw$$

En la forma discreta podemos obtener la energía como la suma de cuadrados de las componentes del espectro de magnitud. Vemos que para el cálculo de la energía no se tienen en cuenta las fases.

2.2.7 Segmentación

El análisis mediante la aplicación STFT da como resultado una secuencia de frames, donde cada una de ellas incorpora información que puede ser usada para realizar la segmentación de la señal, con el fin de extraer la melodía que la compone. En nuestro caso, en cada fragmento se mantiene, entre otras cosas, el valor de energía, y frecuencia fundamental calculado para dicho frame.

Como era de esperar, el framework CLAM ofrece el algoritmo de segmentación que será empleado para la extracción de la melodía, que posteriormente será convertida a mensajes MIDI. Este algoritmo se basa en el cálculo de inicio y fin de cada nota, mediante la observación de las variaciones de pitch y energía entre frames. Con el fin de ofrecer la posibilidad de realizar un ajuste fino sobre los resultados de la segmentación, se incorporará un editor gráfico, que permita realizar modificaciones sobre los límites temporales en cada nota de la melodía obtenida.

2.3 MIDI

Acrónimo de Musical Instrument Digital Interface, el MIDI es un protocolo ideado para comunicar instrumentos musicales entre sí, y también con ordenadores.

Se puede decir que el MIDI es como la “partitura” de una pieza musical registrada, por ejemplo, en un CD. En una partitura tradicional, no sólo se encuentran notas musicales y sus duraciones, sino también simbología de diferentes tipos que indica compás, expresividad, instrumento que realizará un determinado pasaje, etc.

Un sintetizador puede recibir mensajes MIDI en su entrada MIDI IN, y generar sonidos en base a dichos mensajes. Estos sonidos se encuentran almacenados en bancos de sonido disponibles en el sintetizador.

El General MIDI, creado en 1990, es un estándar nacido de la necesidad de poder distribuir archivos MIDI y que estos tengan la mayor similitud posible al ser reproducidos en máquinas diferentes, esto es, si se indica, por ejemplo, que tiene que sonar un determinado instrumento, que sea ese el instrumento y no otro que dependa del fabricante del sintetizador; aunque un

mismo instrumento puede tener mejor o peor calidad de sonido dependiendo de la calidad del sintetizador.

Este estándar también establece unas características mínimas que debe cumplir un sintetizador que sea compatible con General MIDI, entre las cuales se encuentran: capacidad multitímica de 16 canales; polifonía mínima de 24 notas; mapa estándar de 128 programas (instrumentos); caja de ritmos accesible siempre desde el canal 10, con un mapa estándar de 59 sonidos de percusión.

En los siguientes apartados se describe de forma breve los mecanismos utilizados por el protocolo MIDI, relevantes en el contexto que nos ocupa, prestando mayor atención a los aspectos relacionados con este proyecto.

2.3.1 Canales

El protocolo MIDI permite que los mensajes sean enviados a 16 canales distintos, correspondientes a direcciones lógicas, que internamente se numeran de 0 a 15 (0000..1111 en binario). Un dispositivo emisor, normalmente envía por un solo canal a la vez, mientras que un dispositivo receptor, generalmente puede recibir por varios canales de forma simultánea

Cada canal puede ser visto como un instrumento independiente, de esta forma, podrían estar sonando varios instrumentos simultáneamente, cada uno por un canal diferente. Recordemos que el canal 10 está reservado para la percusión.

2.3.2 Mensajes

Existen distintos tipos de mensajes MIDI, cada uno con una duración fija (normalmente dos o tres bytes). Un dispositivo MIDI debe disponer de un microprocesador que sea capaz de entender estos mensajes, aunque tampoco es necesario que los entienda todos; si el dispositivo no entiende un mensaje, simplemente lo ignora y pasa al siguiente.

El primer byte del mensaje es el byte de status, cuyo bit más de mayor peso está siempre a 1; el byte siguiente es el primer byte de datos, y dependiendo del mensaje, puede haber un segundo byte de datos. El bit más significativo en los bytes de datos está siempre a cero.

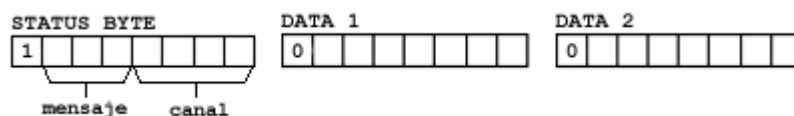


Fig 12. Estructura binaria de un mensaje MIDI.

Como se puede ver en la figura anterior, se tienen 3 bits disponibles para definir el tipo de mensaje, lo cual es indicativo de que existen 8 tipos diferentes de mensajes MIDI. Para indicar el número de canal al cual se enviará el mensaje, se dispone de 4 bits, lo cual coincide con el número de canales soportados por el protocolo. La existencia del segundo byte de datos depende del tipo de mensaje. La siguiente tabla muestra de forma esquemática, los diferentes tipos de mensajes MIDI.

Name	Binary	Hex.	Data 1	Data 2
Note Off	1000 nnnn	8N	pitch	velocity
Note On	1001 nnnn	9N	pitch	velocity
Poly. Aftertouch	1010 nnnn	AN	pitch	pressure
Control Change	1011 nnnn	BN	control type	value
Program Change	1100 nnnn	CN	program	
Chan..Aftertouch	1101 nnnn	DN	pressure	
Pitch Bend	1110 nnnn	EN	MSByte	LSByte
System Message	1111 xxxx	FX		

Algunos mensajes de control (que también son mensajes de canal, ya que son enviados sobre un canal concreto), serán comentados en el apartado siguiente; en este solamente comentaremos algunos de los mensajes de canal restantes más significativos para el contexto del proyecto, y el resto se mencionarán únicamente de pasada.

El mensaje **Note On** se utiliza para indicar al dispositivo que se debe iniciar la reproducción de una nota. El primer byte de datos indica el tono o altura (pitch) de la nota que se desea reproducir, y el segundo byte determina la velocidad (fuerza o intensidad sonora) que debe aplicarse a dicho tono. Como ya se ha comentado anteriormente, el primer bit en los bytes de datos está siempre a cero, por lo que se dispone de 7 bits para datos en cada byte, lo que nos da un rango de valores de 0 a 127, por tanto, tenemos 128 tonos distintos repartidos en escalas de doce tonos (también llamada escala cromática), y sus múltiplos (octavas). Esto nos da una tesitura de más de diez octavas.

El mensaje **Note Off** indica la finalización de una nota. El primer byte de datos indica el pitch, y el segundo la velocidad con la que debe desaparecer dicha nota, por lo que a un mayor valor de velocidad, más bruscamente desaparecerá. En lugar de emplearse este mensaje, los dispositivos emisores suelen generar un mensaje de Note On con velocidad cero.

El mensaje **Pitch Bend** se emplea para aplicar una variación en el pitch de la nota. Aplicar un pequeño porcentaje de desafinación es en ocasiones aconsejable, ya que puede incorporar un mayor realismo, frente a la “perfección” digital. En este mensaje, se combinan los dos bytes de datos para dar un único valor de 14 bits (-8192..+8191). Cuando este valor es nulo no provoca variación alguna. El General MIDI establece que el rango de desafinación debe ser de +/-2 semitonos.

El mensaje **Program Change** se usa para seleccionar el instrumento que debe emplearse, para lo cual disponemos de un byte de datos con 7 bits disponibles, lo cual resulta en 128 instrumentos diferentes. Los instrumentos pueden agruparse en bancos de 128 instrumentos cada uno, por lo que puede seleccionarse el banco que se desea utilizar; para este fin existe un mensaje de control que realiza el cambio de banco.

Los mensajes **Polyphonic Aftertouch** y **Channel Aftertouch** se utilizan para detectar la presión que se ejerce, por ejemplo, sobre las teclas de un teclado.

2.3.3 Control

Este mensaje engloba 128 tipos de control, definidos por el primer byte de datos, de los cuales sólo está asignada una pequeña parte, por lo que existen muchos disponibles para su asignación futura. Algunos de estos mensajes usan el rango de 0 a 127 y otros utilizan valores de encendido/apagado, en este caso puede haber un valor de ON (0..63) y otro de OFF (64..127).

Uno de los mensajes de control más usados es el **Control Change 7**, que corresponde al volumen. El segundo byte indica el valor del control, por lo que el volumen puede ajustarse en un rango de 0 a 127. Los mensajes de cambio de volumen afectan de manera uniforme a todo el canal al cual son enviados, por lo que para hacer que las notas que están sonando en un determinado canal, tengan diferentes intensidades, puede emplearse, por ejemplo, el valor de la velocidad comentado anteriormente, el cual puede ser aplicado a cada nota de manera independiente.

Cuando se comentó el mensaje Program Change, se indicó además que los instrumentos podían ser agrupados en diferentes bancos. El mensaje que indica un cambio de banco es el **Control Change 0**, donde el valor del segundo byte de datos informa sobre el banco deseado.

Un mensaje de control a tener en cuenta es el **Control Change 121: Reset-All Controllers**, el cual restaura los controles restantes a su valor por defecto. Es útil enviar este mensaje a la finalización o inicio de una reproducción, con el fin de asegurar la estabilidad de cara a sucesivas reproducciones. Otro mensaje importante es **Control Change 123: All Notes Off**, que resulta de gran utilidad en caso de quedarse alguna nota “colgada” por alguna causa como que se pierda su respectivo mensaje de Note Off.

Existen otros tipos de mensajes de control; por ejemplo, controles que aplican efectos sonoros como modulación en amplitud (trémolo) o en frecuencia (vibrato), reverberación, chorus, etc.

Los mensajes de sistema, cuyo byte de status empieza con 1111, tienen un comportamiento distinto, ya que los cuatro bits restantes no indican ningún número de canal, sino que afectan de forma global al dispositivo al cual son enviados.

Una vez realizada la segmentación de la señal original, ya tenemos una melodía con información adecuada, disponible para ser convertida en mensajes MIDI. En este apartado se ha querido presentar una descripción básica del protocolo MIDI, con el fin de ofrecer una visión general de su funcionamiento.

3. Requisitos de la aplicación

En todo proyecto de Ingeniería de Software, debe realizarse un análisis previo para estudiar los requerimientos del sistema que se desea implementar, y decidir si vale la pena invertir el correspondiente esfuerzo en su elaboración.

Los requisitos son capacidades y condiciones con las cuales debe ser conforme el sistema. Existen varios tipos de requerimientos, y su cantidad y naturaleza dependen del tipo de aplicación que se quiera construir y del dominio en que se encuentre. En este proyecto dividiremos los requisitos en dos grandes bloques:

- Requisitos funcionales:
- Requisitos no funcionales.

En este capítulo se presentan los distintos tipos de requerimientos para el Sistema de Conversión de Voz a MIDI, y finalmente se hace una valoración sobre la viabilidad del proyecto. Debe tenerse en cuenta que un estudio sobre los costes en proyectos reales en diferentes empresas, reveló que 37% de ellos están relacionados con los requisitos; de forma que las cuestiones sobre requisitos constituyen la principal causa de problemas.

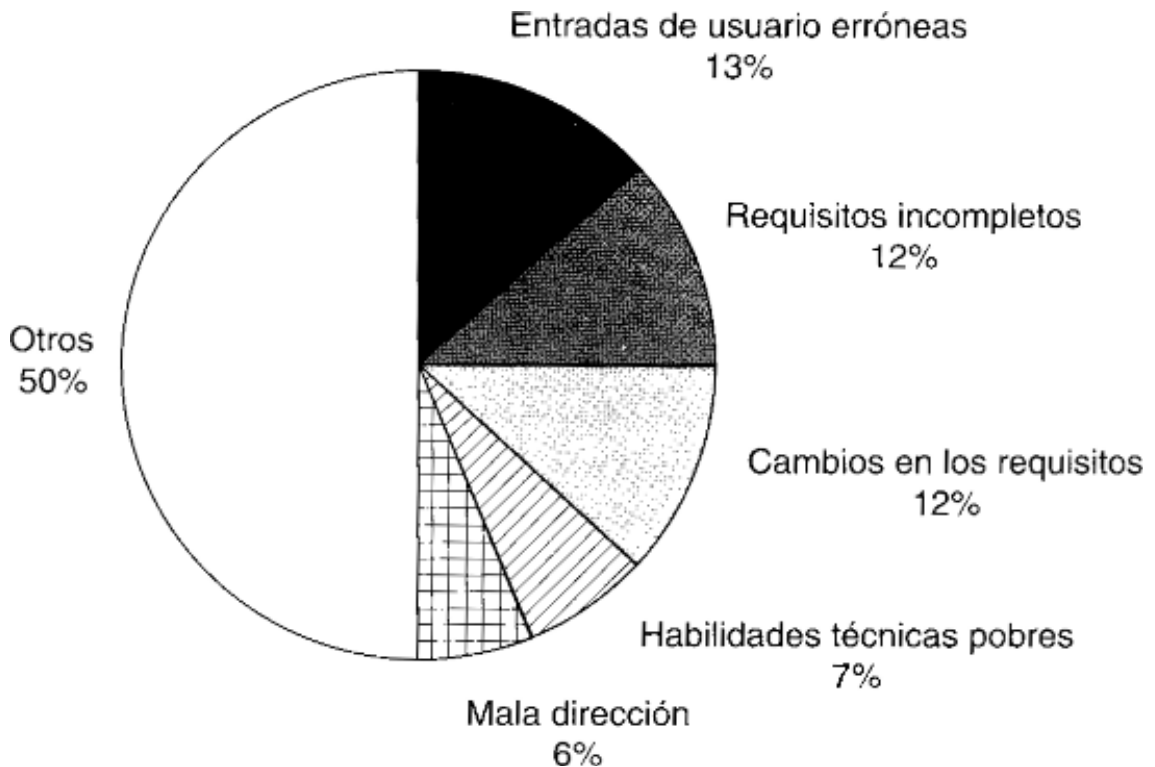


Fig 13. Costes en proyectos software

Cabe destacar que aunque dada la estructura de esta memoria, podría parecer que se ha aplicado un ciclo de vida en cascada, en realidad se trata de un desarrollo iterativo, donde en cada iteración se introducen nuevas funcionalidades, y se refinan los anteriores en caso de que sea necesario.

3.1 Requisitos funcionales

Los requisitos funcionales son las características que ofrecerá el sistema, independientemente de la forma en que éstas sean implementadas. Para el Sistema de Conversión de Voz a MIDI se han identificado los siguientes requisitos funcionales:

- **Captura de señal de audio.** La aplicación será capaz de capturar señales de audio y mantenerlas en memoria para su posterior procesamiento.

- **Reproducción de audio**. El programa permitirá la reproducción de archivos de audio, de forma que el usuario pueda escuchar el audio original y el resultado obtenido después del análisis y la segmentación (F0 en el tiempo y melodía obtenida).
- **Capacidad para cargar y guardar archivos de audio**. El sistema ofrecerá al usuario los mecanismos necesarios para cargar archivos de audio, así como la posibilidad de almacenar archivos previamente cargados o capturados.
- **Obtención de las características necesarias en el proceso de conversión**. Se incorporarán los medios adecuados para realizar un análisis sobre el audio, que permita extraer la información requerida por el proceso de conversión.
- **Segmentación**. El sistema realizará una segmentación de la señal, con el fin de obtener la secuencia de notas que forma su melodía.
- **Interfaz gráfica de usuario (GUI)**. Las funcionalidades del sistema serán ofrecidas al usuario a través de una interfaz gráfica.
- **Visualización y edición de la información**. El sistema permitirá al usuario la posibilidad de visualizar y editar la información obtenida ofreciendo para ello diferentes vistas.
 - *Vista de captura, reproducción y señal original*: El usuario podrá obtener una representación de la forma de onda de la señal original, así como una vista dinámica de la misma durante la captura y reproducción de audio.
 - *Vista de análisis*: Una vez realizado el análisis de la señal, se dispondrá de la opción de ver los resultados obtenidos, y consultar valores de amplitud, energía y F0 de la señal, en cualquier instante de tiempo de forma interactiva.
 - *Vista de edición de la segmentación*: Una vez extraída la melodía, deberá ser posible editar los límites temporales de las distintas notas que la forman, con el fin de tener la posibilidad de hacer un ajuste fino en caso de que sea necesario.
 - *Piano Roll*: Se ofrecerá una vista tipo pianola, en la cual se mostrarán los resultados de la segmentación con sus valores MIDI vs. valores originales (amplitud, frecuencia y energía). La pianola también permitirá la edición de los límites temporales en las notas de la melodía para su posible ajuste.
- **Conversión a mensajes MIDI**. El sistema será capaz de realizar un proceso de conversión a mensajes MIDI a partir de la información contenida en la melodía extraída.
- **Reproducción MIDI**. Se añadirán los mecanismos necesarios para permitir la reproducción de los mensajes obtenidos en la conversión MIDI, así como la posibilidad de elegir entre varios instrumentos disponibles.
- **Almacenamiento del resultado**. El sistema tendrá la capacidad necesaria para el almacenamiento de los resultados obtenidos en la conversión.

Una forma conveniente de representar los requisitos funcionales, es mediante un diagrama de casos de uso. Los diagramas de casos de uso ofrecen una visión general de alto nivel sobre las funcionalidades del sistema, lo cual es útil en el desarrollo, ya que pueden consultarse en cualquier momento durante la gestión de requisitos.

Además de los diagramas, también se elaboran descripciones detalladas de todos los casos de uso, utilizando lenguaje estructurado. Para este fin, se diseñan una serie de plantillas que describen con detalle y paso a paso todo aquello que sucede durante el transcurso de la ejecución de cada caso de uso. Sin embargo, en esta memoria, se presentará únicamente el diagrama de casos de uso general, ya que se piensa que no es necesario añadir aquí la descripción detallada de todos los casos de uso implicados en la aplicación.

El diagrama de casos de uso para el Sistema de Conversión de Voz a MIDI se presenta en la página siguiente.

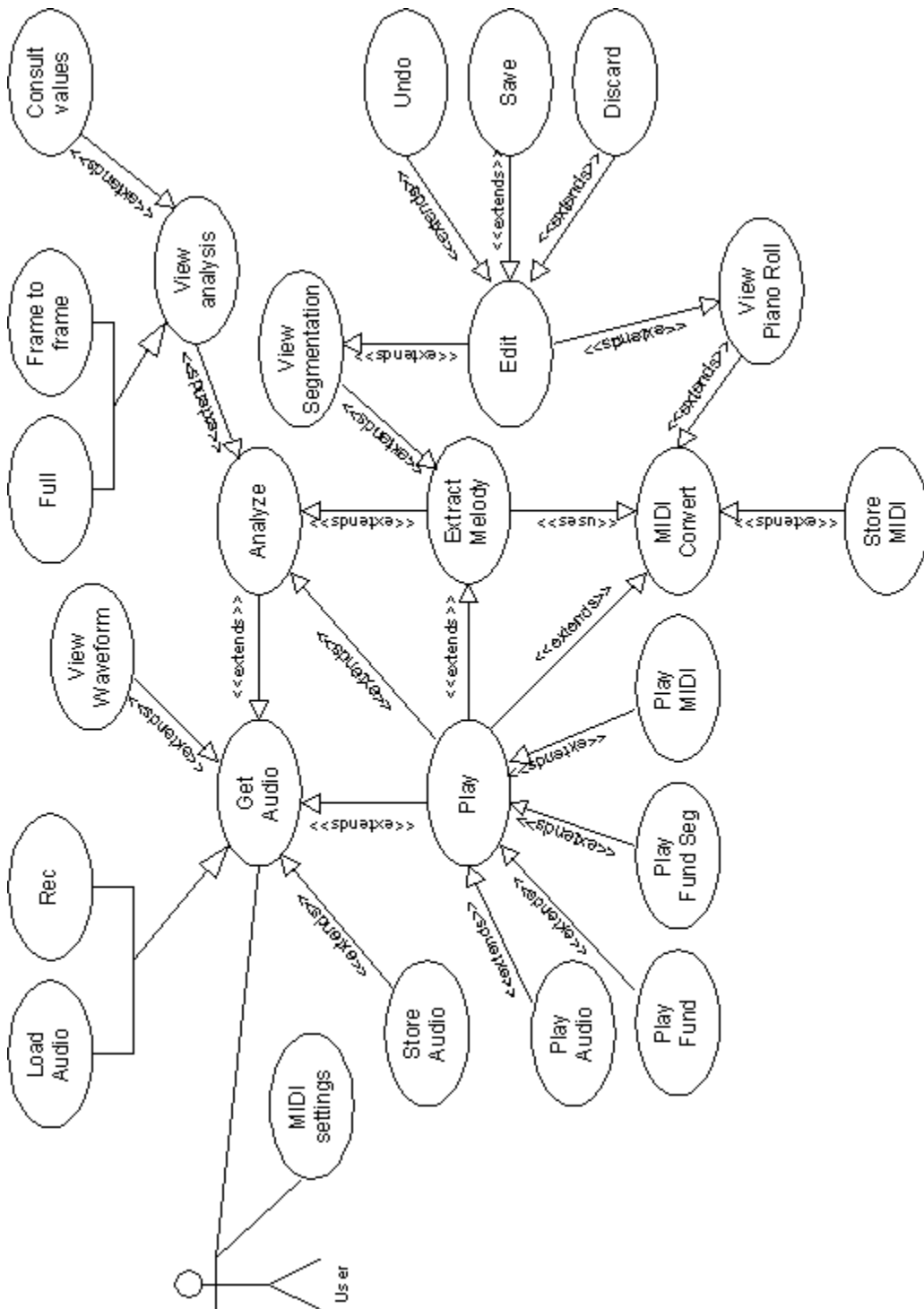


Fig 14. Diagrama general de casos de uso para el Sistema de Conversión de Voz a MIDI.

En el diagrama de casos de uso quedan reflejadas las distintas funcionalidades que ofrecerá el sistema. Podemos ver, por ejemplo, que cuando un usuario carga o captura audio, entonces tiene la opción de reproducirlo o realizar un análisis de la señal. Asimismo, cuando hay un análisis hecho, se pueden visualizar y consultar los resultados obtenidos, y reproducir la frecuencia fundamental de la señal a lo largo del tiempo. También se observa, que una vez hecho el análisis, se tiene la oportunidad de extraer la melodía, lo cual realiza la conversión a MIDI (uses MIDI Convert); entonces, el usuario tiene la opción de guardar el archivo MIDI correspondiente o editar la melodía o la melodía MIDI obtenidas tras la segmentación, así como reproducir la melodía o el MIDI correspondiente. Para más detalles sobre la realización de los casos de uso, se pueden consultar los escenarios de casos de uso situados en el anexo de la memoria.

3.2 Requisitos no funcionales

Los requisitos no funcionales constituyen las restricciones a las cuales estará condicionada la elaboración y funcionamiento del sistema, tales como herramientas que se usarán en la elaboración, lenguaje de programación empleado, etc. Para este proyecto han sido identificados los siguientes requisitos no funcionales:

- La extracción de las características necesarias para la conversión se llevará a cabo utilizando técnicas de análisis espectral.
- Los parámetros empleados para la segmentación serán la energía y la frecuencia fundamental de la señal (observando sus variaciones en el tiempo).
- El lenguaje de programación utilizado será C++.
- El núcleo del sistema será implementado bajo el entorno CLAM: C++ Library for Audio and Music.
- La aplicación trabajará con archivos de audio en formato wave.
- La captura de la señal se hará con una velocidad de muestreo de 11025 Hz y una resolución de 16 bits.
- La interfaz de usuario se construirá usando el kit de herramientas Qt.
- Para la visualización de los datos, se usará el API (Application Program Interface) OpenGL, haciendo uso de la interfaz que ofrece Qt para permitir su integración.
- La interfaz ofrecerá al usuario una forma cómoda e intuitiva para la utilización del sistema.

Los requerimientos presentados, constituyen una representación de las operaciones que el usuario podrá llevar a cabo mediante el uso de la aplicación; esto no quiere decir que durante la elaboración y construcción del programa no puedan haber variaciones y matices sobre los requisitos; de hecho, tal como hemos indicado anteriormente, se trata de un desarrollo iterativo en el cual, durante la elaboración del producto, pueden haber cambios o modificaciones en los requisitos que favorezcan una refinación progresiva del sistema.

3.3 Valoración de la viabilidad del sistema

Una vez identificados los requisitos iniciales para el Sistema de Conversión de Voz a MIDI, estamos en disposición de hacer una valoración para determinar si el sistema es factible, y decidir si se realiza un estudio serio y una elaboración completa del mismo.

En este caso, la decisión está condicionada, entre otras cosas, por el entorno de trabajo. Aunque la discusión sobre las herramientas empleadas se hará posteriormente, éstas son de

interés relevante en lo que respecta a la decisión de si se continúa o no con el desarrollo, dado que deben ofrecer los servicios necesarios para llevar a cabo la implementación del sistema.

Tal como se ha indicado anteriormente, el núcleo de la aplicación, es decir, todos los aspectos relacionados con el procesamiento del audio y gestión del MIDI, se llevarán a cabo haciendo uso del framework CLAM. Después de hacer un estudio sobre las características de CLAM, y en concreto, de las que están más relacionadas con las necesidades de este proyecto, se ha decidido que el framework ofrece las herramientas y mecanismos adecuados para su implementación, por tanto, en este aspecto se da el visto bueno en la continuidad del desarrollo.

Otra cuestión a tener en cuenta está relacionada con la construcción de la interfaz de usuario, y su integración con el sistema. El toolkit gráfico empleado debe cumplir con los requerimientos solicitados por la aplicación, y permitir la separación del interfaz respecto del sistema, es decir, la interfaz gráfica de usuario no tiene por qué conocer la existencia del sistema para poder ofrecer sus servicios al usuario.

El toolkit gráfico Qt cumple con las necesidades requeridas por la aplicación, e incorpora el soporte para la integración de OpenGL, que será el API empleado para la visualización de datos, por lo que en este caso no hay inconveniente para llevar a cabo el proyecto.

El equipo de desarrollo, formado por el director y tutor del proyecto, y por mi mismo, está de acuerdo en su realización, por tanto, en este punto se decide que el sistema es viable, que puede cumplirse con la fecha de entrega, inicialmente en Junio de 2004, y que será implementado en base a los requisitos iniciales presentados anteriormente.

4. Herramientas empleadas

En este capítulo, se comentan las herramientas más destacadas, que se han utilizado para el desarrollo del programa. En primer lugar, se hace una discusión sobre las características y capacidades del lenguaje C++, creado por Bjarne Stroustrup, y de las ventajas de su uso en relación a este proyecto. A continuación se mencionan los compiladores empleados para la construcción de la aplicación. Seguidamente se presenta una descripción breve del framework CLAM, enfocada según las necesidades del desarrollo que nos ocupa, y para finalizar, se comentan de forma escueta las características de Qt, el kit de herramientas gráficas utilizado en el desarrollo de la interfaz gráfica de usuario.

4.1 C++

El lenguaje C++ fue desarrollado a partir del lenguaje C, por lo que C++ es un superconjunto respecto de C, que retiene a C como subconjunto, por lo tanto, las librerías de C se pueden utilizar desde un programa en C++, y la mayoría de las herramientas que dan apoyo a la programación en C pueden ser usadas en C++. El nombre indica la naturaleza evolutiva de los cambios a partir de C (“++” es el operador de incremento de C); y como nota curiosa comentaremos que los expertos de la semántica de C determinaron que C++ es inferior a ++C.

C++ puede utilizar las mismas llamadas a función y secuencias de retorno que C (u otras más eficientes). Uno de los objetivos originales de C era reemplazar el código de ensamblador para las tareas de programación de sistemas con más demanda, por lo que C es un lenguaje flexible y válido para la escritura de código a bajo nivel; cuando se diseñó C++ se tuvo especial cuidado para no comprometer los beneficios en este área.

La principal diferencia que existe entre C y C++ es el concepto de clase que incorpora C++. Este diseño facilita la estructuración racional de programas grandes, por lo que el mantenimiento resulta más sencillo y una sola persona puede hacerse cargo de mayores cantidades de código. Además, este enfoque permite la ocultación de dicho código detrás de interfaces elegantes.

Durante los años siguientes a la primera aparición de C++, denominado entonces “C con Clases”, se fueron incorporando nuevas mejoras en la definición del lenguaje, como el manejo de excepciones, la identificación de tipos en tiempo de ejecución, los espacios de nombres (namespaces), el uso de plantillas (templates), resolución de la sobrecarga, miembros protected, herencia múltiple, etc.

La utilidad de las plantillas, se desarrolló principalmente para dar soporte a los contenedores tipados estáticamente (listas, vectores, mapas...), y para dar apoyo a un uso elegante y eficiente de dichos contenedores (programación genérica). La biblioteca estándar de C++, definida en el espacio de nombres *std* ofrece una rica colección de contenedores e iteradores, denominada armazón STL (Standard Template Library), por lo que el programador tiene disponible un importante grupo de estructuras de uso común en la escritura de programas, sin necesidad de tener que implementarlas desde cero.

Como se puede deducir de lo dicho hasta ahora, C++ es un lenguaje orientado a objetos. Las características principales de un lenguaje orientado a objetos son: encapsulación, herencia y polimorfismo. Tal como hemos comentado anteriormente, el concepto clave de C++ es la clase. Una clase es un tipo definido por el usuario. Las clases ofrecen ocultación de datos (encapsulación), inicialización garantizada de datos, conversión implícita de tipos para los tipos definidos por el usuario, tipos dinámicos, gestión de la memoria controlada por el usuario, y mecanismos de sobrecarga de operadores. C++ retiene la habilidad de C para tratar eficientemente los objetos fundamentales del hardware (bits, bytes, palabras, direcciones, etc.), lo que permite que los tipos definidos por el usuario puedan ser implementados con un alto grado de eficiencia. Por otro lado, el lenguaje permite la coexistencia de fragmentos de código

programado en otros lenguajes en el interior del código en C++, lo cual lo hace todavía más valioso, flexible y eficaz.

El concepto de encapsulación (ocultación de datos) es importante. Las clases de C++ permiten la encapsulación de datos a distintos niveles: `public`, `protected` y `private`. De esta forma, el usuario puede tener un control preciso sobre el acceso a los datos definidos en el interior de la clase que está implementando.

```
class MyClass
{
public:
    // declaraciones visibles para todas las clases.
protected:
    // declaraciones visibles para la propia clase
    // y para sus clases derivadas, y no accesibles
    // para el resto de clases.
private:
    // declaraciones visibles únicamente por la propia clase
    // e inaccesibles para el resto de clases, incluidas las
    // clases derivadas.
};
```

El control de acceso a miembros de la clase puede hacerse por medio de métodos declarados en la sección pública. De esta forma, la clase puede tener miembros y métodos que realizan operaciones interiores a la clase y que no interesa que puedan ser manipulados por otras clases, con el fin de proteger la integridad y consistencia de dichas operaciones, que únicamente atañen a la propia clase. Este concepto sería de difícil implementación en un lenguaje no orientado a objetos como C o Pascal, dado que estos lenguajes no soportan las tecnologías orientadas a objetos, por lo que emular este concepto mediante estos lenguajes sería innecesariamente laborioso y traumático para el programador.

Otro aspecto no menos importante lo constituyen la inicialización segura de los miembros (variables, objetos, etc), definidos en la clase. En un lenguaje no orientado a objetos deberían definirse funciones, que podrían llamarse por ejemplo `create(...)`, que realizaran la inicialización de los datos de un tipo definido por el usuario, mediante el uso de `typedef` en caso de tratarse del lenguaje C. Para el caso de una lista doblemente enlazada cuyos elementos fueran valores enteros, esto podría ser implementado en C de la siguiente forma:

```
typedef struct node{
    int item;
    struct node *next;
    struct node *prior;
}NODE;

typedef struct{
    int count;
    NODE *head;
}DBLINK;

/* crear lista vacia */
void create(DBLINK *db)
{
    db->count = 0;        /* cero elementos */
    db->head = NULL;     /* vacia -> cabecera == NULL */
}
```

que puede utilizarse:

```
DBLINK *dbLink;
create(dbLink);
```

Este tipo de inicialización puede ser peligrosa, ya que el programador podría olvidar hacer la llamada a la función create. Las clases de C++ proporcionan una inicialización segura mediante el uso de constructores. Un constructor es un método que tiene el mismo nombre que la clase

```
class MyClass
{
public:
    MyClass();
};
MyClass::MyClass()
{
    // inicializaciones
}
```

En C++, como sucede en otros lenguajes orientados a objetos, se permite la sobrecarga de constructores, por lo que una clase puede tener disponible más de un constructor para su inicialización, dependiendo de las necesidades del programador.

```
MyClass(int);
MyClass(int, bool f=false, double d=0.0);
MyClass(MyClass&);
...
```

Como podemos ver, también se permite el uso de parámetros por defecto, lo cual también es aplicable a métodos propios de la clase que no son constructores; en este caso, los parámetros por defecto deben colocarse al final de la signatura. El constructor de la clase es llamado cuando se hace una instancia a un objeto de la clase, lo cual asegura la inicialización segura de dicha instancia:

```
MyClass myObject0();
MyClass myObject1(3);
MyClass myObject2(myObject0);
```

La destrucción de los objetos también se hace de forma controlada por medio de los destructores, que son llamados cuando el objeto es eliminado. Los destructores permiten la liberación de los recursos utilizados por la instancia, que ya no son necesarios, ya que dicha instancia ha sido eliminada. Los destructores no pueden ser sobrecargados y se declaran con el mismo nombre de la clase precedidos del símbolo ~. La definición de nuestra clase de ejemplo con constructores y destructor quedaría de la siguiente forma:

```
class MyClass
{
public:
    // constructores
    MyClass();
    MyClass(int, bool f=false, double d=0.0);
    MyClass(MyClass&);

    // destructor
    ~MyClass();
};
MyClass::MyClass()
{
    // inicializaciones
}
MyClass::MyClass(int i, bool f, double d)
{
    // inicializaciones
}
```

```

MyClass::MyClass(MyClass& myObject)
{
    // constructor de copia
    // la instancia se inicializa
    // con los datos del parámetro
}

MyClass::~MyClass()
{
    // destructor
    // liberar recursos empleados por la instancia
}

```

Un lenguaje de programación sirve para dos propósitos relacionados: sirve como vehículo para que el programador especifique las acciones a ejecutar, y ofrece un conjunto de conceptos para que el programador los utilice cuando piense en lo que debe hacerse. La primera finalidad, requiere idealmente de un lenguaje que esté “cerca de la máquina”, de forma que se traten los aspectos importantes de manera simple y eficiente, de modo que resulte razonablemente evidente para el programador; el lenguaje C se diseñó teniendo principalmente en cuenta esto. El segundo propósito, requiere idealmente de un lenguaje que se encuentre “cerca del problema a resolver”, de modo que los conceptos de una solución puedan ser expresados de forma directa y concisa; las utilidades añadidas a C en la creación de C++ fueron diseñadas teniendo principalmente esto en cuenta.

Una de las herramientas más poderosas para gestionar la complejidad es la organización jerárquica, es decir, la organización de los conceptos en una estructura de árbol con el concepto más general como raíz. En C++, las clases derivadas representan dichas estructuras, por lo que un programa puede organizarse con frecuencia como un conjunto de árboles o grafos acíclicos dirigidos de clases; esto es, el programador especifica un número de clases base, cada una con su propio conjunto de clases derivadas. Las funciones virtuales, usadas para la implementación del concepto de polimorfismo, se pueden utilizar a menudo para definir operaciones para la versión más general de un concepto (clase base), y cuando sea necesario, se puede refinar la interpretación de estas operaciones con el fin de adaptarlas a casos particulares especiales (clases derivadas). En C++ la herencia puede ser implementada de la siguiente forma:

```

MyClassB
{
    // definición de la clase base
};

MyClassD : public MyClassB
{
    // la clase derivada posee los datos y miembros
    // definidos en la clase base.

    // definiciones de la clase derivada
};

```

Cabe destacar, que el motivo principal en torno al cual gira la elección del lenguaje C++, para la implementación de la aplicación que será desarrollada durante este proyecto, puede describirse de forma clara: tanto el entorno CLAM como el toolkit Qt, herramientas fundamentales en el desarrollo del programa, están implementadas en C++, de modo que dadas las capacidades que ofrece este lenguaje, y su implicación directa respecto al ámbito del proyecto, C++ se considera la mejor elección como lenguaje de programación a emplear en su desarrollo.

4.2 Compiladores

Aunque esta memoria no tiene la finalidad de describir las características de funcionamiento de un compilador, sí se cree conveniente al menos, presentar de forma básica su estructura y las fases que tienen lugar en el proceso de compilación y enlace, durante la construcción de un archivo ejecutable.

A grandes rasgos, un compilador es un programa que lee un programa escrito en un determinado lenguaje (el lenguaje fuente), y lo traduce en un programa equivalente escrito en otro lenguaje (el lenguaje objeto); como parte importante de este proceso de traducción, el compilador informa al usuario de la presencia de errores en el programa fuente.

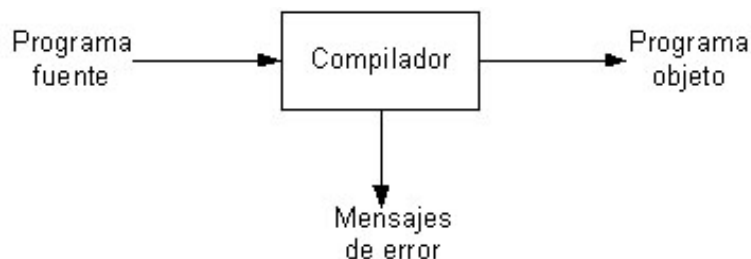


Fig 15. Un compilador.

Además de un compilador, pueden ser necesarios otros programas para llevar a cabo la creación de un programa ejecutable. Un programa fuente se puede dividir en módulos almacenados en archivos distintos. La tarea de reunir el programa fuente, a menudo se confía a un programa distinto, llamado preprocesador. El preprocesador también puede expandir abreviaturas, llamadas macros, a proposiciones del lenguaje fuente.

El programa objeto creado por el compilador puede requerir procesamiento adicional antes de poderlo ejecutar. En la figura siguiente se muestra un esquema, donde se realiza, en primer lugar un preprocesamiento, y a continuación el compilador crea código en lenguaje ensamblador, el cual es traducido por un ensamblador a código de máquina, y después se enlaza a algunas rutinas de biblioteca con el fin de producir el código que realmente se ejecutará en la máquina.

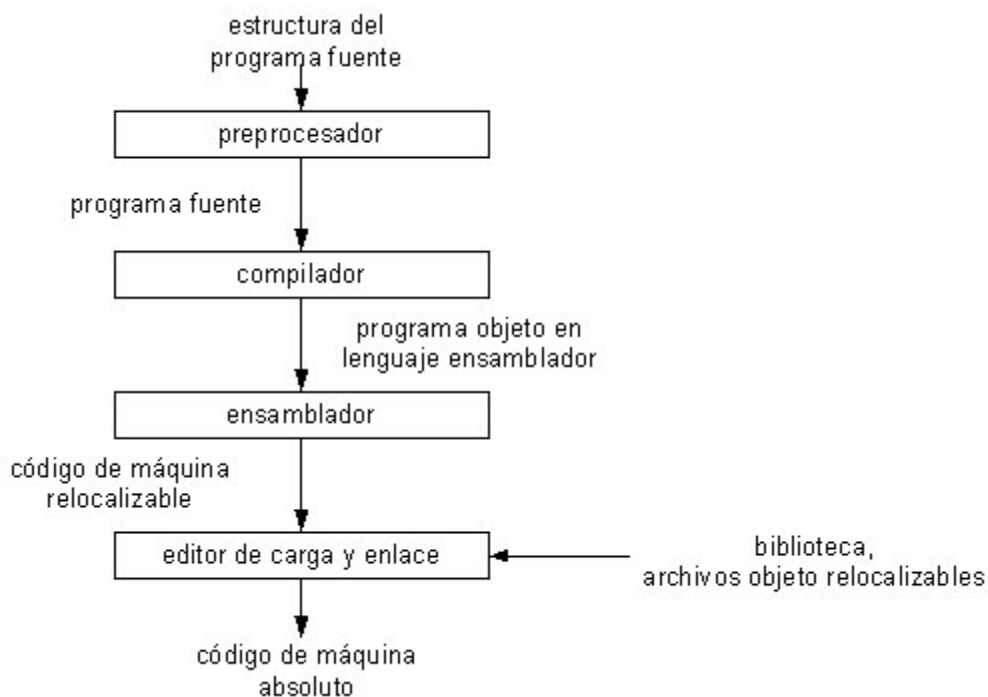


Fig 16. Sistema para el procesamiento de un programa escrito en algún lenguaje.

Ahora que ya hemos descrito de forma básica el funcionamiento de un compilador, pasaremos a mencionar los compiladores que se han utilizado en la construcción del programa desarrollado durante este proyecto, de forma igualmente breve. De hecho, no vamos a describir aquí las características y detalle de los compiladores, sino que simplemente haremos una mención hacia ellos, dado que el proceso de compilación ya ha sido descrito.

Dada la naturaleza multiplataforma de CLAM, y las posibilidades que ofrece Qt se ha pensado que sería interesante que la aplicación pudiera ser construida tanto en la plataforma Linux como en Windows, disponiendo del mismo código fuente. En este aspecto, el framework CLAM no ofrece problema alguno, pero sí que existe un pequeño matiz respecto de Qt. El problema se encuentra en que Qt no es GPL (o por lo menos no totalmente), y solo tiene licencia libre para desarrollar bajo Linux; no obstante, Trolltech ofrece una edición no comercial de Qt para Windows, por lo que en esta plataforma se ha empleado esta versión de Qt durante la implementación.

Como era de esperar, esta edición no comercial de Qt distribuida libremente por Trolltech, no se corresponde con la última versión existente de Qt, sí disponible bajo Linux, pero para la implementación de la aplicación que nos ocupa será suficiente, y el código igualmente válido para ser compilado bajo Linux con la última versión; a este respecto, comentar que la edición no comercial corresponde a la versión 2.3 y la disponible en Linux es la 3.1.1-62. ¿Por qué se comenta ahora todo esto sobre Qt? La razón está directamente relacionada con la compilación del programa y se aclarará inmediatamente.

Como ya hemos dicho, el programa podrá ser construido tanto en Windows como en Linux. Para Linux, la distribución utilizada es SuSE 8.2, y se utilizan los compiladores gcc y g++ de GNU en su versión 3.3. En el caso de Windows, el compilador usado es el Visual C++ 6.0. Ahora pasaremos a aclarar el tema referente a Qt.

Entre los servicios que ofrece el framework CLAM, se encuentra un entorno para la gestión de proyectos y construcción del producto final realmente útil para el desarrollo. Haciendo uso de este entorno, el programador puede definir todos los parámetros necesarios para la construcción del programa de forma sencilla. Como Qt no tiene licencia de software libre bajo Windows, entonces bajo este entorno, todos los parámetros necesarios para Qt deben ser añadidos “a mano”, a las propiedades del archivo de proyecto creado por el entorno de CLAM, hecho que no sucede cuando se construye bajo Linux. Aunque esto, en principio no supone un problema complejo, se comenta por creerse necesario hacerlo.

Ya que CLAM, nos ofrece este entorno con todo lo necesario para la construcción del programa de forma cómoda, podemos imaginar que esta será la herramienta utilizada como mediadora en el proceso de compilación. Además, no hacerlo sería poco recomendable. Los detalles para llevar a cabo la construcción de la aplicación a partir del código fuente se citan en el anexo.

4.3 CLAM: C++ Library for Audio and Music

En este apartado haremos una descripción básica del framework CLAM, y de su implicación en el desarrollo de este proyecto. El título de este proyecto bien podría haberse extendido a ‘Sistema de Conversión de Voz a MIDI *mediante el uso de CLAM*’, ya que es precisamente este entorno de desarrollo el que da soporte para la realización de las tareas esenciales en el programa.

CLAM es un framework de libre distribución con licencia GNU-GPL, que permite el desarrollo de completas aplicaciones de audio con carácter multiplataforma, mediante el uso de C++ y a través de avanzados algoritmos de procesamiento. CLAM no solo puede dar soporte en la parte relativa a procesamiento en una aplicación, sino que también puede ayudar en otros aspectos como:

- Acceso a dispositivos de audio y MIDI.
- Gestión de hilos de ejecución (threads).

- Serialización de objetos en formatos como XML y SDIF.
- Visualización y control sobre los datos.
- Integración de la visualización mediante el uso de diversos toolkits.
- Interconexión de los módulos de la aplicación de forma independiente...

CLAM también está capacitado para realizar procesos complejos como:

- Tratamiento de datos heterogéneos: no solo muestras, sino también información espectral, símbolos, información estructurada...
- Flujos de datos complejos: con eventos asíncronos (controles), diferentes mecanismos de alimentación...
- Escalabilidad mediante composición de procesos pequeños.
- Creación dinámica e interconexión de redes de procesos...

Además, también ofrece herramientas que disponen de algoritmos preparados para realizar:

- Modelado espectral y transformaciones.
- Extracción de características (especialmente útil para este proyecto).
- ...

CLAM fue desarrollado como proyecto interno en el MTG (Music Technology Group), en la UPF (Universitat Pompeu Fabra), e inicialmente llamado *MTG-Classes*. El proyecto fue creado como una librería de clases en C++ pensada para ser usada en los proyectos de investigación internos del MTG, y las metas fueron ofrecer un entorno flexible, completo e independiente de la plataforma, para análisis y síntesis de sonido escrita en C++, que cubriera las necesidades presentes y futuras en todos los proyectos del MTG.

Los tres ejes principales para alcanzar estas metas fueron:

- **Compleitud:** debe incluir todas las herramientas que puedan ser necesarias para la implementación de un proyecto sobre procesamiento de audio (E/S, procesos, almacenamiento, visualización...)
- **Flexible:** De uso sencillo y adaptable a todo tipo de necesidades.
- **Independiente de la Plataforma:** Compilable bajo plataformas UNIX, Windows y Macintosh.

Estos objetivos iniciales han ido cambiando paulatinamente desde la etapa inicial, para acomodarse al hecho de que la librería no fuera únicamente una herramienta interna para el MTG, sino también pública bajo GNU-GPL. Cualquier persona que disponga de conexión a internet, puede visitar la página de CLAM y descargar el código fuente del framework completo, así como las librerías externas requeridas por CLAM, necesarias para su correcto funcionamiento, y toda la documentación relativa a la instalación y uso de este completo entorno de trabajo para procesamiento de audio, que por otra parte va creciendo día a día e incorporando nuevas capacidades en este área.

El principal elemento en CLAM es el proceso. Un proceso tiene asociada una clase para su configuración, y unos datos para procesar mediante la aplicación de algún algoritmo. Los procesos se inician, ejecutan y detienen por medio de métodos comunes en todos ellos, ya que todos derivan de la clase *Processing*, en la cual se implementan dichos métodos, los cuales pueden ser modificados o implementados totalmente por el usuario según sea su naturaleza. Los procesos suelen trabajar con datos de proceso (*ProcessingData* en CLAM).

Los *ProcessingData* son clases de objetos que pueden encapsular atributos, y los métodos asociados para su correcta manipulación. Una característica relevante y nada usual que ofrece CLAM la constituyen los atributos dinámicos. Un *ProcessingData* se define en CLAM como un tipo dinámico, el cual posee atributos dinámicos. Estos atributos tienen la ventaja de poderse instanciar y desinstanciar en tiempo de ejecución, lo cual permite, entre otras cosas, un uso coherente de la memoria, al poder iniciar un objeto únicamente con los atributos correspondientes a las propiedades necesarias para realizar la tarea deseada, o añadir y eliminar atributos en tiempo de ejecución. Además de la gran cantidad de clases *ProcessingData* que

ofrece CLAM, preparadas para cubrir las necesidades de cualquier aplicación de procesamiento de audio (es poco común pensar en la necesidad de alguna clase de este tipo, y descubrir que no está disponible ya en CLAM), el usuario también puede definir las suyas propias, lo cual le otorga un alto grado de escalabilidad.

Como ya hemos comentado antes, cualquier clase de proceso en CLAM deriva de la clase `Processing`, y normalmente tiene asociada una clase para su configuración con el mismo nombre de la clase de proceso, añadiendo el sufijo `Config`. Esto supone una forma conveniente para inicializar los datos del proceso antes de comenzar su ejecución. Es muy común, que un proceso esté a su vez compuesto de procesos. En CLAM también se define una clase adecuada para estos casos llamada `ProcessingComposite`, cuyo nombre indica que se trata de un proceso que ejecuta a su vez algún determinado número de procesos en su interior.

Para mantener control sobre los procesos en ejecución, se dispone de clases de control que permiten alterar el comportamiento del proceso. Por ejemplo, podríamos pensar en un oscilador, controlado por una clase de control que le envía de forma asíncrona el valor de frecuencia a la que debe producirse la oscilación.

El lo que se refiere a las necesidades del Sistema de Conversión de Voz a MIDI, el entorno CLAM ofrece todos los servicios necesarios para su implementación, dado que incorpora clases de objetos adecuadas para la realización de todas las tareas propias de la aplicación. Tenemos disponible una clase proceso generador de ventanas de distintos tipos, para aplicarlas en el proceso STFT, comentado en el capítulo sobre conceptos fundamentales en el apartado de análisis espectral, así como un proceso para realizar la FFT sobre la señal de entrada enventanada, lo que nos permite obtener su espectro; procesos para la detección de picos espectrales, los cuales servirán de entrada al proceso, también disponible, para la detección de tono fundamental (pitch), métodos para el cálculo de la energía, procesos para realizar la segmentación con el fin de obtener la melodía correspondiente a la entrada de audio...

En la parte de acceso a dispositivos de audio y MIDI, el framework CLAM también dispone de los mecanismos adecuados para llevar a cabo las tareas implicadas, como la reproducción y el registro. En relación a esta funcionalidad, comentar que en las fechas correspondientes al desarrollo de este proyecto, el framework no incorporaba soporte para la salida MIDI, dado que en esos momentos CLAM daba soporte únicamente a la entrada MIDI. Este tema fue estudiado e implementado, precisamente para ofrecer soporte de salida MIDI de carácter multiplataforma, con el fin de que este proyecto pudiera ser implementado totalmente haciendo uso de CLAM. A este respecto, agradecer la colaboración prestada a Maarten de Boer, miembro del MTG y del equipo de desarrollo de CLAM, gracias a cuya intervención esto ha sido posible. En relación a la implementación de la interfaz de dispositivo de salida MIDI para CLAM bajo la plataforma MS-Windows, yo mismo me encargué de realizar las modificaciones adecuadas en la definición del dispositivo de entrada ya presente en CLAM, con el fin de que pudiera dar soporte también a la salida de MIDI, a través de la interfaz ofrecida por la librería `portmidi`, integrada en el entorno CLAM como recurso de librería externa.

En el caso de la captura de audio, también hubo un pequeño problema referente al soporte para el audio de entrada ofrecido por CLAM bajo la plataforma Linux, a través del interfaz ofrecido por la arquitectura de sonido ALSA (Advanced Linux Sound Architecture), que fue corregido adecuadamente, de forma que la captura de audio puede ser realizada actualmente sin problemas bajo ambas plataformas. Por tanto, puede decirse que el desarrollo de este proyecto no solamente ha sido útil respecto de la implementación de la aplicación en sí, sino que también ha servido para depurar e incorporar mejoras en el framework CLAM, que como hemos comentado anteriormente, se encuentra en constante desarrollo y evolución, publicándose nuevas versiones del entorno que incorporan nuevas características y funcionalidades, y mejoran las ya existentes en caso de ser necesario, todo ello sin perjudicar el código escrito para versiones anteriores del entorno.

En lo referente a la gestión del almacenamiento, CLAM ofrece clases para facilitar la carga y almacenamiento de archivos en distintos tipos de formato, como wav, mp3, ogg... Aunque para este proyecto, inicialmente se utilizará únicamente el formato wav en relación al audio.

Por todo lo expuesto, es claro que la elección de CLAM como herramienta principal para el desarrollo de este proyecto es una decisión acertada, y más cuando imaginamos toda la cantidad de horas de trabajo que supondría la implementación desde cero de toda esta cantidad de funcionalidades, y pretender además que sean multiplataforma como lo es CLAM.

Como hemos comentado en un apartado anterior, CLAM también ofrece un entorno que proporciona los mecanismos necesarios para la gestión de proyectos y construcción del producto final. El usuario únicamente tiene que definir los parámetros que necesita su aplicación en un archivo de propiedades (settings.cfg), y CLAM se encarga del resto. Esta característica simplifica enormemente la gestión de proyectos, y supone para el usuario el no tener que preocuparse ni emplear tiempo en este aspecto, lo cual sin duda ahorra muchas horas de trabajo.

El equipo de desarrollo de CLAM, también ha pensado en facilitar el aprendizaje del uso del framework, poniendo a disposición de quien lo desee, toda la documentación, la cual puede ser descargada directamente desde la web de CLAM o ser consultada en línea, así como un gran número de ejemplos de código completos, que ilustran el uso de las clases que forman la librería. Todos estos ejemplos se encuentran ya preparados para ser compilados y ejecutados después de la instalación de CLAM. Y por si todo esto fuera poco, también se encuentra disponible un tutorial (el cual yo mismo he realizado), que permite al usuario que lo completa con éxito, tener un conocimiento inicial de CLAM bastante sólido, como base a futuros desarrollos más avanzados que puede realizar posteriormente.

4.4 Qt: el toolkit gráfico de Trolltech

Qt es una librería de clases en C++, y un kit de herramientas para interfaces gráficas de usuario (GUI), para sistemas Windows y Unix, entre los que se encuentra Linux que también es un sistema basado en Unix.

Los kits de desarrollo de interfaces gráficas de usuario, no son bien conocidos en el mundo Macintosh y MS-Windows, pero sí omnipresentes en sistemas Unix. Esto es porque el API de Windows y las herramientas de desarrollo de Macintosh, ya contienen mecanismos de alto nivel para facilitar la construcción de elementos gráficos, como botones y barras de desplazamiento, y también funciones para la manipulación de colores, fuentes, y otros tipos de componentes visuales.

El sistema de ventanas predominante en sistemas Unix es el X-Window System, el cual es muy flexible, pero no ofrece mucha ayuda al programador. Lo que ofrece es un conjunto de primitivas que permiten el dibujo de elementos como líneas y rectángulos, asignar color de fondo, color frontal, y manejar algunas interacciones de usuario y otros eventos. No hay nada que permita la creación de botones o barras de desplazamiento, y menos todavía elementos más complejos como cuadros de diálogo o barras de tareas.

Evidentemente, nadie quiere programar una aplicación completa bajo estas condiciones; es por esto que se han ideado los toolkits gráficos con el fin de facilitar el desarrollo GUI bajo sistemas Unix. Existen muchos toolkits gráficos para elegir a la hora de desarrollar bajo Unix, y probablemente el más conocido es Motif, porque es el que ha sido adoptado por la mayoría de vendedores de sistemas Unix, como toolkit nativo para desarrollo GUI.

El CDE (Common Desktop Environment), que corre en algunos sistemas basados en Unix, como Solaris y HP-UX, está basado en Motif. Sin embargo, muchos programadores coinciden en que el uso de Motif es complejo y no demasiado divertido, y que los programas escritos con Motif son mucho más largos que los escritos con Qt que realizan la misma tarea.

En lo referente a sistemas Windows, bien es cierto que el API de Windows contiene funciones que permiten la creación de elementos GUI, y la manipulación de fuentes colores y demás. Usar este conjunto de funciones, es indudablemente más sencillo que programar directamente sobre el X-Window System, pero continúa siendo engorroso para la realización de trabajos reales. Las herramientas que ofrece Windows como la conocida MFC (Microsoft

Foundation Classes), tampoco son demasiado sencillas de utilizar e insuficientes para el desarrollo de interfaces gráficas sofisticadas, por lo que los programadores emplean demasiado tiempo en la construcción de la interfaz, en lugar de concentrarse en las necesidades reales de la aplicación que están desarrollando. Precisamente aquí es donde cobran una mayor importancia los frameworks como Qt.

Qt facilita la creación e inicialización sencilla para una completa colección de elementos, comúnmente necesarios en la elaboración de una interfaz de usuario gráfica, como botones o elementos de menú, y una fácil manipulación de los mismos para capturar las interacciones de usuario. Además, incorpora la definición de aspectos importantes como relaciones padre-hijo y también relaciones geométricas entre objetos. Trabajar con Qt, permite al programador centrarse al máximo en el desarrollo de la aplicación que está creando.

Otro aspecto interesante para la elección de Qt, lo constituye su carácter multiplataforma, ya que el mismo código escrito puede ser recompilado y ejecutado en diversas plataformas, lo cual no restringe a la aplicación desarrollada a poder correr solamente bajo un único sistema operativo. Por otra parte Qt, a diferencia de otros toolkits gráficos que utilizan tecnología basada en capas, emplea directamente las primitivas gráficas de la plataforma en la que se encuentra, lo cual desemboca en una mayor rapidez de ejecución. Los sistemas de capas que usan otros toolkits, implican la realización de las llamadas a través de una capa situada encima del API nativo; esto tiene la ventaja de que el toolkit es relativamente sencillo de implementar, y se asegura la correspondencia en lo referente al aspecto gráfico propio del estilo de la plataforma en cuestión, pero posee el inconveniente de que la ejecución es más lenta, ya que puede ser necesario que la orden deba atravesar varias capas, con el correspondiente retardo que ello comporta.

Qt explota muchas de las características de C++, dado que éste es su lenguaje de implementación. Por supuesto hace uso de clases y herencia, también usa polimorfismo mediante la utilización de funciones virtuales, sobrecarga de operadores, acceso a funciones miembro y uso de plantillas. Qt no hace uso de espacios de nombres, identificación de tipos en tiempo de ejecución (RTTI), u otras características nuevas de C++ no soportadas todavía por todos los compiladores; tampoco utiliza la colección de contenedores STL, pero el programador sí puede usarla en sus propios desarrollos.

Como característica innovadora en lo referente a la comunicación entre objetos, Qt utiliza un sofisticado sistema de señales y ranuras (signals and slots). Mediante este sistema, las notificaciones y paso de información entre los objetos que forman la interfaz gráfica de usuario, y las interacciones entre los mismos, se realiza de forma sencilla, eficaz y realmente cómoda para el programador.

Por supuesto, existen otros toolkits gráficos, que además tienen la ventaja de tener licencia GPL en todas las plataformas. Uno de ellos es FLTK, bajo el cual se sustenta el CLAMVM (CLAM Visualization Module). Precisamente, el carácter de libre uso ofrecido por este toolkit, fue el motivo por el cual el equipo de desarrollo de CLAM optó por utilizarlo en la implementación del módulo de visualización de CLAM. Este módulo de visualización está implementado de forma independiente al entorno, por lo que su utilización es opcional. El soporte para Qt ofrecido por CLAM se encuentra en desarrollo, y se espera incorporar características al respecto en futuras versiones del framework.

No pongo en duda de que el desarrollo de la interfaz de usuario del Sistema de Conversión de Voz a MIDI, podría haberse realizado completamente haciendo uso de FLTK, pero existen varias razones por las cuales se ha preferido hacerlo con Qt. El toolkit Qt ofrece al programador una rica colección de widgets (componentes gráficos), que pueden cubrir prácticamente cualquier necesidad imaginable, aspecto más limitado en FLTK. El aspecto de los widgets es más agradable en Qt que en FLTK; esto podría considerarse un factor secundario, pero la realidad es que al usuario común le llama más la atención una aplicación cuyo aspecto sea más estéticamente correcto que otra que ofrezca la misma funcionalidad y cuyo aspecto sea menos agradable a la vista. Otra razón de vital importancia se encuentra en la facilidad de implementación. Aunque FLTK también incorpora soporte para OpenGL y otras características

que hubieran facilitado la construcción del interfaz, su uso es relativamente complejo y poco intuitivo. A todo esto hay que añadir que la documentación ofrecida para el uso de FLTK es más bien escasa y que no hay ejemplos de código que ilustren el uso de los widgets, tal como ocurre en el caso de Qt, que ofrece una documentación muy completa y ejemplos de código disponibles para ser compilados y ejecutados.

Por este motivo, y todos los expuestos anteriormente, en este apartado dedicado a la descripción breve y general de este toolkit, se considera que la elección de Qt, constituye la mejor elección respecto del toolkit gráfico a emplear para el desarrollo de la interfaz gráfica de usuario en este proyecto.

5. Diseño e implementación

En este capítulo se describen los métodos empleados en el diseño e implementación del software que satisface los requerimientos presentados en el capítulo 3 (Requisitos de la aplicación). Se empieza con una introducción dedicada a exponer el concepto de diseño orientado a objetos (diseño OO). En los siguientes apartados se describen de forma esquemática las ideas y criterios utilizados para el diseño e implementación de cada una de las partes del sistema, así como la forma en que se comunican entre sí. También se dedica un apartado en el cual se comenta el sistema de ayuda de la aplicación.

5.1 Introducción

Las técnicas de diseño orientado a objetos son esenciales para la creación de software de fácil mantenimiento, robusto y bien definido, haciendo uso de las tecnologías y lenguajes orientados a objetos como C++ o Java. Sin embargo, conocer un lenguaje orientado a objetos como C++, aunque necesario, es un primer paso insuficiente para crear sistemas de objetos; para ello es también necesario conocer como “pensar en objetos”.

¿Cómo deberíamos asignar las responsabilidades a las clases de objetos? ¿Cómo deberían interactuar los objetos? ¿Qué clases deberían hacer qué? Estas son preguntas clave en el diseño de un sistema. Por otro lado, la experiencia acumulada por los desarrolladores orientados a objetos, y demás desarrolladores software, ha contribuido de forma significativa para la creación de un repertorio, tanto de principios generales como de soluciones basadas en la aplicación de ciertos estilos, que ofrecen una guía importante para la creación de software. Estos principios y estilos, han sido codificados de forma estructurada, dándole un nombre a cada uno de ellos con el fin de asociar de forma rápida un problema con una solución, abstrayendo todo ello en dicho nombre. Son los denominados **Patrones Software**, transmitidos y empleados ampliamente por los diseñadores de sistemas OO. Con el fin de ilustrar esta idea, se presenta el siguiente patrón de muestra:

Nombre del patrón:	Experto en Información.
Solución:	Asignar una responsabilidad a la clase que tiene la información necesaria para cumplirla.
Problema que resuelve:	¿Cuál es el principio básico mediante el cual asignaremos responsabilidades a los objetos?

Existen varios catálogos de patrones, cada uno de los cuales se especializa en un área concreta del desarrollo software. El patrón de muestra forma parte del catálogo de patrones GRASP (**G**eneral **R**esponsibility **A**ssignment **S**oftware **P**atterns) propuesto por Craig Larman. Los patrones GRASP describen un conjunto de ideas y conceptos básicos y esenciales, que deben tenerse en cuenta en todo diseño de objetos. Otro grupo de patrones de vital importancia es el catálogo GoF (**G**ang **o**f **F**our), especializado en el diseño de objetos, y que consta de 23 patrones descritos en el libro Design Patterns; este catálogo debe su nombre (la pandilla de los cuatro) al hecho de haber sido escrito por cuatro autores. En el catálogo POSA (**P**atterns-**O**riented **S**oftware **A**rchitecture), se describen patrones orientados al diseño arquitectural de sistemas software, entre los cuales se encuentra el patrón Capas (Layers).

Aparte de los catálogos de patrones mencionados, también existen patrones aplicados a otras áreas de desarrollo software como son: Patrones de Análisis, los cuales se aplican durante el análisis de requisitos como apoyo para el diseño del modelo del dominio. Patrones para el diseño de interfaces de usuario. Patrones de pruebas.

También necesitaremos un lenguaje adecuado para el análisis y diseño orientado a objetos, así como para representar los “planos del software”, que nos sirva como herramienta para pensar, y como forma de comunicación con otros. En relación con este aspecto, es interesante

aplicar **UML** (Unified Modeling Language) al servicio del A/DOO.

El Lenguaje de Modelado Unificado (UML), es un lenguaje para especificar, visualizar, construir y documentar los artefactos de los sistemas software, así como para el modelo del negocio y otros sistemas no software. UML se ha convertido en el estándar de facto para el modelado orientado a objetos, comenzando como una iniciativa de Grady Booch y Jim Rumbaugh en 1994, para combinar las notaciones visuales de sus dos populares métodos: los métodos de Booch y OMT (Object Modeling Technique). Más adelante, se les unió Ivar Jacobson, creador del método Objctory, y el grupo comenzó a ser conocido como *los tres amigos*. Muchos otros contribuyeron a UML, entre los que cabe destacar a Cris Kobryn, que lidera el proceso de refinamiento de UML que todavía continúa. UML fue adoptado en 1997 como estándar por el OMG (Object Management Group), que promueve estándares para la industria.

El lenguaje UML ya ha sido empleado anteriormente para representar los casos de uso, mediante el diagrama de casos de uso, presentado en el capítulo dedicado a discutir sobre los requisitos de la aplicación.

Durante el análisis de requisitos, se puso énfasis en la investigación del problema y en las funcionalidades que el sistema ofrecería al usuario, en vez de ponerlo en una solución. El diseño, pone énfasis en la descripción de una solución conceptual que debe satisfacer los requisitos identificados durante el análisis, en lugar de ponerlo en la implementación. Por tanto, análisis y diseño pueden ser definidos con frases como *“hacer lo correcto”* (análisis) y *“hacerlo correcto”* (diseño). Por último, durante la implementación, los objetos de diseño se implementan como clases en un lenguaje de programación orientado a objetos como C++.

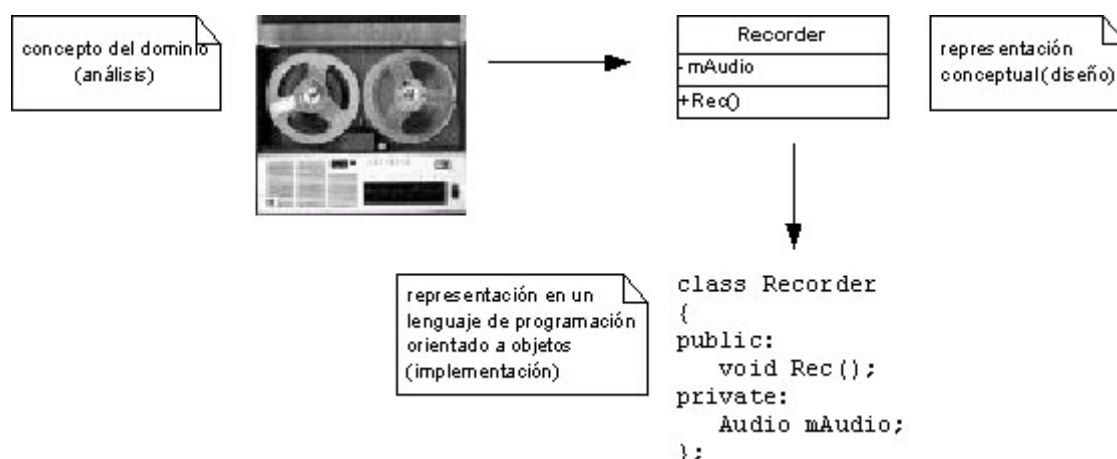


Fig 17. Uso de la orientación a objetos para su representación.

El diagrama de la figura 17 puede servir de ejemplo para ilustrar los conceptos de análisis, diseño e implementación de un sistema software. Existen diversos modos de representar una idea o concepto por medio de un diagrama UML:

- Diagrama de casos de uso.
- Diagrama del dominio.
- Diagrama de clases.
- Diagrama de secuencia.
- Diagrama de colaboración.
- Diagrama de estado.
- Diagrama de actividades.
- Diagrama de despliegue.

En los siguientes apartados de este capítulo, se hará uso de alguno de estos tipos de diagrama como apoyo visual que facilite la comprensión de las ideas y criterios empleados. De todos

modos, a continuación se muestra un diagrama que expresa una idea simplificada para el modelo del dominio en el sistema que nos ocupa.

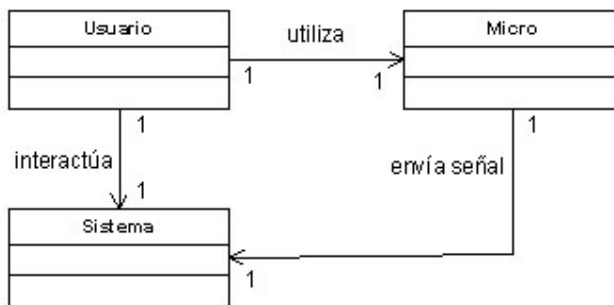


Fig 18. Modelo simplificado del dominio.

El modelo del dominio no es una descripción de los objetos software, sino una visualización de los conceptos en el mundo real.

También resulta útil crear una vista estática de las definiciones de las clases mediante un diagrama de clases de diseño, que a diferencia del modelo de dominio, no muestra conceptos del mundo real, sino clases software.

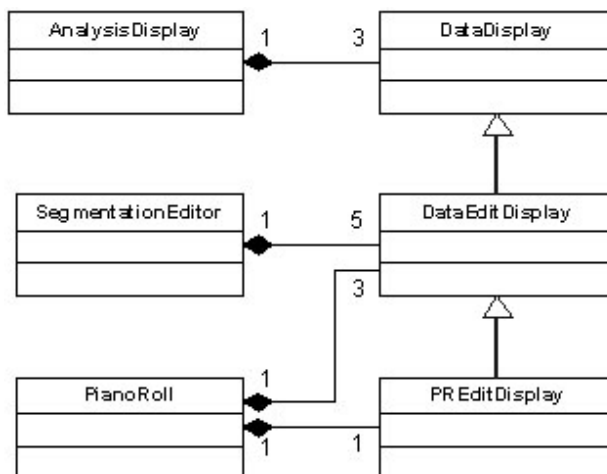


Fig 19. Diagrama de clases de diseño para el sistema de visualización.

El diagrama de clases de la figura 19 muestra de forma esquemática el diseño para la implementación de los distintos modos de visualización que ofrece el Sistema de Conversión de Voz a MIDI. Este diagrama, así como los anteriores presentados en este apartado dedicado a la discusión sobre el diseño orientado a objetos, han sido presentados como apoyo a la exposición de este concepto. En los siguientes apartados de este capítulo, se explicará con un poco más de detalle lo referente al diseño e implementación de las diferentes partes de las que consta el sistema.

5.2 Arquitectura interna

En este apartado se describen las diferentes partes de las que consta el núcleo del sistema, y a través de las cuales se realizan los procesos principales de la aplicación. Podemos hacer una separación bien definida de la arquitectura interna en tres partes:

- E/S de audio.
- Análisis y segmentación.
- Salida MIDI.

5.2.1 E/S de audio

El Sistema de Conversión de Voz a MIDI debe cumplir con los siguientes requisitos de entrada/salida de audio:

- Carga de archivos de audio en formato wave.
- Almacenamiento de archivos de audio en formato wave.
- Captura de audio mediante micrófono.
- Reproducción de audio.
 - Reproducción del audio original.
 - Reproducción de frecuencia fundamental contenida en el audio a lo largo del tiempo.
 - Reproducción de la frecuencia fundamental después de la segmentación (melodía).

Observando estos requerimientos, nos damos cuenta de que la aplicación debe ofrecer soporte completo tanto para la entrada como para la salida de audio, debiendo acceder a dispositivos que permitan la reproducción y captura de audio, así como controlar el acceso a disco para lectura y escritura de archivos.

Como ya hemos comentado anteriormente, el framework CLAM ofrece soporte completo para E/S de audio, ya que incorpora las clases necesarias para realizar los procesos requeridos, así como las estructuras con las cuales trabajan dichos procesos. Las clases de las que hablamos son clases **Processing**, y las estructuras que trabajan con ella son clases **ProcessingData**.

Para llevar a cabo las funcionalidades que satisfacen todos los requisitos de E/S de audio para el Sistema de Conversión de Voz a MIDI, asignaremos responsabilidades a las siguientes clases, cada una de las cuales se encargará de realizar una tarea concreta tal como se enumera a continuación:

- `AudioFileIO`
 - función: cargar y almacenar archivos de audio en formato wave.
- `AudioPlayer`
 - función: reproducir audio original.
- `FundPlayer`
 - función: reproducir la frecuencia fundamental contenida en el audio original.
- `FundSegPlayer`
 - función: reproducir la frecuencia fundamental obtenida en el proceso de segmentación; es decir, la serie de notas que forman la melodía.
- `AudioRecorder`
 - función: capturar una señal de audio.

Asignar una tarea concreta a cada clase favorece un **Bajo Acoplamiento**, dado que cada objeto se encargará por sí mismo de realizar su trabajo sin depender de la existencia de otros. Además de esta ventaja, este enfoque permite una mayor facilidad en el mantenimiento del código y en la identificación de errores. Por otro lado, este conjunto de clases, las cuales constituyen el motor de entrada/salida de audio de la aplicación, forma un módulo con **Alta Cohesión**, ya que cada objeto se encarga de realizar una tarea concreta, y entre todos dan soporte completo para la E/S de audio del programa. Estos criterios constituyen principios básicos que deben ser tenidos en cuenta en el desarrollo de todo sistema orientado a objetos, tal como se ha comentado en el apartado anterior cuando se hizo mención a los **Patrones GRASP**, y serán continuamente empleados durante todo el diseño del sistema. A continuación, describiremos de forma breve y conceptual el funcionamiento interno de cada clase. Para entrar en más detalle sobre la implementación, se pueden examinar los archivos de código fuente incluidos en el CD que acompaña al proyecto.

Gestión de disco

Para la implementación de la clase `AudioFileIO`, se han empleado las clases `MonoAudioFileReader` y `MonoAudioFileWriter` ambas derivadas de la clase

Processing. Estas clases que ofrece CLAM para lectura/escritura de archivos de audio son de un manejo realmente sencillo, y facilitan enormemente la labor al programador. Para la configuración se utilizan las clases `MonoAudioFileReaderConfig` y `MonoAudioFileWriterConfig` respectivamente, derivada de `ProcessingConfig`. La clase de configuración permite determinar con precisión el comportamiento que tendrán los objetos. En nuestro caso, configuraremos los objetos de forma que trabajen con archivos en formato wave. La clase `AudioFileIO`, tiene dos métodos, los cuales se encargan de la carga y del almacenamiento del audio.

```
/* cargar archivo de audio */  
void Load(const char* fileName,Audio& out);  
  
/* almacenar un archivo de audio en formato wav */  
void Save(const char* fileName,Audio& in);
```

Tanto el método `Load` como el método `Save`, reciben como parámetro el nombre del archivo y un objeto de la clase `Audio` (derivada de `ProcessingData`). En el caso del método `Load`, se devuelve por referencia el audio cargándolo en el objeto `out` de la clase `Audio`. El método `Save` toma el audio que será guardado en disco mediante el parámetro `in`. En el interior de ambos métodos se configuran los objetos de las clases `MonoAudioFileReader` y `MonoAudioFileWriter` (derivadas de `Processing`) que se encargarán de cargar y almacenar el audio a través de los objetos `out` e `in` respectivamente.

Reproducción y captura

Para satisfacer los requerimientos de reproducción y captura de audio del sistema, emplearemos como parte principal, objetos de la clase `AudioOut` y `AudioIn`, ofrecidos por CLAM y derivados de la clase `Processing`.

La entrada/salida de audio en CLAM está gestionada por la clase `AudioManager`, la cual también es un proceso y debe iniciarse antes de que entren en acción las clases `AudioIn` o `AudioOut` según sea el caso. El manejador de audio se encarga, entre otras cosas, de asignar la velocidad de muestreo y el tamaño de cada frame que trabajará con el dispositivo de entrada o salida.

Además de esto, hay un factor importante a tener en cuenta: los procesos de reproducción y captura de audio deben ser lanzados como hilos de ejecución (threads), para dar ocasión al usuario de detener la reproducción o grabación en curso, lo cual es de vital importancia en el caso de la captura, ya que de lo contrario no se tendría ninguna opción que permitiera terminar el proceso de grabación.

En el caso de la captura de audio, se inicializa el manejador con una velocidad de muestreo de 11025 Hz y un tamaño de frame de 2048 muestras. El objeto de la clase `AudioIn`, utilizado para obtener el audio, se configura como una entrada para un sólo canal (mono). Una vez configurados e iniciados el manejador de audio y el canal de entrada, se entra en un bucle donde se obtienen los frames de audio mediante el canal de entrada (objeto de la clase `AudioIn`), y se van colocando en un objeto de la clase `Audio`. El procedimiento de grabación continuará hasta que sea detenido por un evento externo; en este caso lo normal es que sea detenido por el usuario.

Todos los métodos y objetos relacionados con la captura de audio se encuentran encapsulados en la clase **`AudioRecorder`**, la cual también incorpora un método para servir el audio registrado al cliente que lo demande.

Para la reproducción de audio, deben satisfacerse requisitos que permitan tres modos de reproducción: audio original, frecuencia fundamental y frecuencia fundamental de cada nota de la melodía obtenida tras el proceso de segmentación. El método principal de cada modo de reproducción es diferente. Para el caso de la reproducción de audio original, se inicializa el manejador de audio y un objeto de la clase `AudioOut`, usado como canal de salida, de forma similar a como se hace en el caso de la captura. A continuación, se entra en un bucle donde se

realiza la fragmentación del audio original, y el envío secuencial de los diferentes trozos, para ser reproducidos a través del proceso que está siendo ejecutado por el objeto de la clase `AudioOut`.

En el caso de la reproducción de la frecuencia fundamental y de la melodía obtenida tras la segmentación, además del manejador de audio y del canal de salida (objeto de la clase `AudioOut`), se utiliza un oscilador para generar la senoide pura correspondiente a la frecuencia deseada en cada momento. CLAM incorpora clases para cubrir estas necesidades; en nuestro caso haremos uso de la clase `SimpleOscillator`, que al ser un proceso también deriva de `Processing`, y tiene asociada su correspondiente clase de configuración `SimpleOscillatorConfig`.

La clase `SimpleOscillator` es controlada a través de clases control. En este caso, obtendremos el control adecuado con el fin de controlar la frecuencia de oscilación que nos interesa en cada momento durante el curso de la reproducción.

Al igual que en el caso de la reproducción de audio original, se configuran e inicializan el manejador y el canal de salida. Además de esto, se configura e inicializa el oscilador, del cual se obtiene el control adecuado con el fin de controlar la frecuencia de oscilación. A continuación se entra en un bucle en el cual se realiza todo el proceso.

En el caso de la reproducción de la frecuencia fundamental, los datos necesarios son la frecuencia contenida en los frames resultantes del análisis, los cuales se mantienen en un objeto de la clase `Frame`, la cual deriva de `ProcessingData`, y que a su vez forma parte del segmento usado como estructura para mantener los datos, es decir, un objeto de la clase `Segment` que también deriva de `ProcessingData`.

Los datos utilizados en la reproducción de la melodía obtenida después del proceso de segmentación, son mantenidos en una estructura melodía, para lo cual se utiliza un objeto de la clase `Melody`, derivada de `ProcessingData`. La melodía que será reproducida es servida por un objeto de la clase `MelodyAnalyzer`, la cual se comentará en el siguiente apartado. El objeto de la clase `Melody`, entre otras cosas, mantiene toda la información que necesitamos para la reproducción, es decir, la lista de notas que forman la melodía. La información de cada una de las notas se encuentra en un objeto de la clase `Note`, la cual deriva de `ProcessingData`. Para el caso de la reproducción, la información que utilizaremos será la relativa a la frecuencia fundamental y a los instantes de tiempo inicial y final de cada nota.

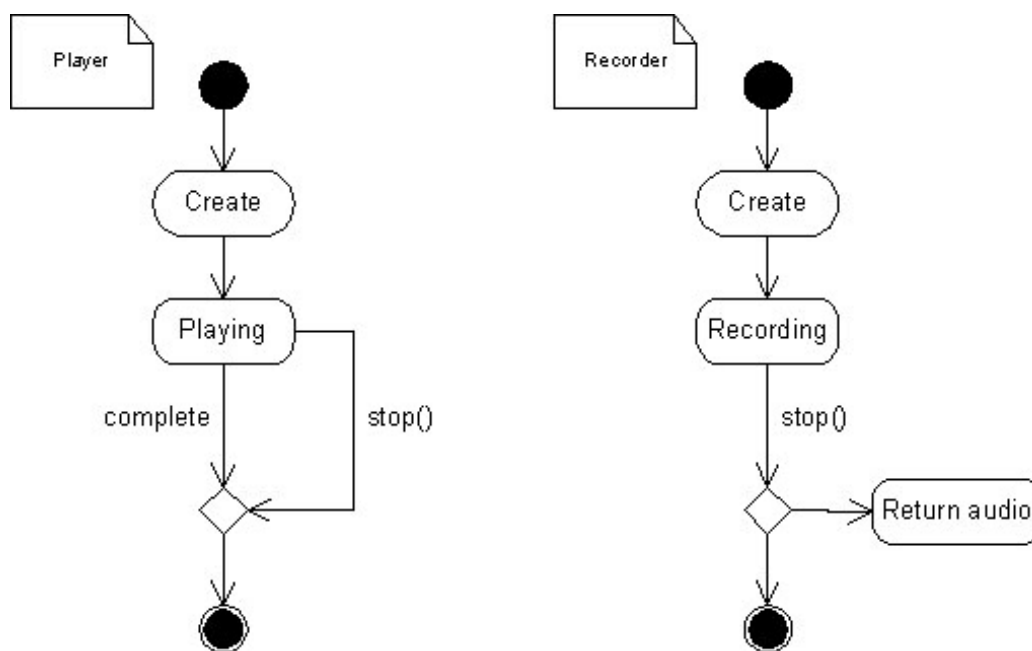


Fig 20. Diagrama de estados simplificado para la reproducción y captura de audio.

En el diagrama de estados de la figura 20, se muestra de forma esquemática el funcionamiento de los procesos de reproducción y captura de audio. Dado que estos procesos son lanzados como hilos de ejecución, tal como se ha comentado anteriormente, se ha optado por iniciar el proceso al instanciar el objeto, es decir, el objeto se crea e inmediatamente después comienza la ejecución del proceso, y cuando el proceso finaliza, se destruye también el objeto. Entonces, los datos necesarios para cada caso se deben pasar como parámetros al constructor de cada objeto.

Las clases que encapsulan los métodos y toda la información necesaria para realizar estas tareas son, tal como se ha comentado anteriormente:

AudioPlayer: reproducción de audio original.

FundPlayer: reproducción de la frecuencia fundamental obtenida en el análisis.

FundSegPlayer: reproducción de la melodía (f0 tras el proceso de segmentación).

AudioRecorder: captura de audio.

A continuación se muestra un diagrama estático con el fin de representar el conjunto de clases que formarán el módulo de entrada/salida de audio para el Sistema de Conversión de Voz a MIDI. Nótese el modo en que las clases activas, es decir, las que tienen su propio hilo de ejecución, pueden ser representadas mediante el lenguaje UML.



Fig 21. Diagrama de clases simplificado para E/S de audio.

En el diagrama de clases de la figura, se han representado los conceptos más relevantes, con el fin de favorecer la claridad, dado que indicar todos los métodos y atributos de las clases en un solo diagrama, desembocaría en un diagrama complejo y de difícil lectura. En el caso de las clases de este módulo, la implementación ha resultado relativamente sencilla y de código bastante escueto, pero en otros módulos, como por ejemplo los de visualización, entran en juego bastantes métodos y atributos como para representarlos por entero y de forma clara en un espacio reducido; por consiguiente, tanto en este capítulo como en los posteriores se muestra una representación algo simplificada con el fin de favorecer su interpretación.

5.2.2 El analizador y el segmentador

La función principal del sistema, es obtener una representación MIDI de la melodía contenida en la señal de audio de entrada; para ello se debe realizar, en primer lugar, un análisis sobre dicha señal, con el fin de extraer las características que nos permitan alcanzar el objetivo deseado.

Tal como se ha indicado anteriormente, la extracción de características contenidas en el audio de entrada, se llevará a cabo haciendo uso de técnicas de análisis espectral; por tanto, una de las cosas que debe hacerse, es obtener el espectro de la señal, para lo cual se hará una implementación de la STFT (ver 2.2.3). A partir de la información espectral obtenida mediante la STFT, haremos una estimación del tono fundamental y de la energía de cada frame. El diagrama de la siguiente figura muestra este concepto de forma esquemática.

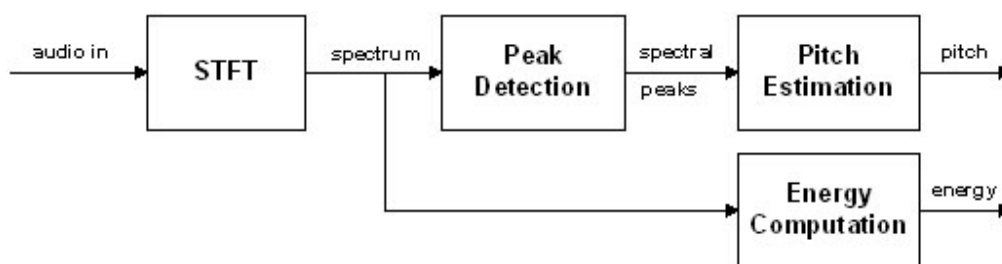


Fig 22. Extracción de características mediante análisis.

Para realizar esta tarea es deseable diseñar, en primer lugar, una clase cuya responsabilidad sea la de obtener el espectro de un frame de audio. Recordemos que la STFT impone una fragmentación del audio de entrada en una secuencia de ventanas (frames), para cada una de las cuales se calcula su espectro. Cada frame de esta secuencia, será enviado al proceso analizador, con tal de obtener los resultados buscados. Entonces, es dentro de este proceso de análisis donde actuará un objeto de la clase `MyAnalyzer`, que se encargará de obtener el espectro del frame, para a continuación obtener las características deseadas a partir de éste.

La responsabilidad de obtener el espectro de cada frame se encomienda a la clase `MyAnalyzer`. El framework CLAM, dispone de los mecanismos necesarios para realizar esta tarea, ofreciendo generadores de ventanas, proceso FFT, y todos los elementos requeridos para facilitar la obtención del espectro. Dado que en esta tarea entran en juego varios procesos, sería adecuado disponer de una clase que se encargara de supervisarlos a todos en conjunto. En CLAM se dispone de la clase `ProcessingComposite`, de la que haremos uso para este fin.

La clase `ProcessingComposite` sigue las reglas del patrón de software `Composite`, y en este caso es encargada de la gestión de sus procesos hijo. Sabemos que un proceso en CLAM debe configurarse, iniciarse, ejecutarse y finalmente detenerse; y que para iniciar un proceso usamos el método `Start`, se ejecuta a través de un método `Do` y que finaliza haciendo una llamada a su método `Stop`. Al derivar nuestra clase de `ProcessingComposite`, podemos hacer todas estas tareas como si se tratara de un solo proceso, en lugar de tener que arrancar de forma explícita cada uno de ellos, es decir, nosotros declaramos una clase `MyAnalyzer` derivada de `ProcessingComposite`, la cual alberga en su interior todos los objetos de proceso necesarios para realizar la labor (procesos hijo); entonces configuramos los procesos hijo en el interior de esta clase compuesta de procesos, y únicamente debemos hacer una llamada a `Start`, `Do` y `Stop` para un objeto de la clase `MyAnalyzer`, en lugar de tener que hacerlo para cada uno de sus procesos hijo.

Ahora que tenemos el analizador espectral, necesitamos una clase que lo utilice y que se encargue de obtener la frecuencia fundamental y la energía de cada frame de audio a partir de su espectro. Dado que para este fin necesitamos otros procesos, nada nos impide derivar otra clase de `ProcessingComposite` para asignarle la tarea. Llamaremos a esta clase `VoiceAnalyzer`, y le asignaremos la responsabilidad de obtener las características que necesitamos, contando

para ello con la colaboración de un objeto de la clase `MyAnalyzer`. Además del proceso realizado por el objeto de la clase `MyAnalyzer` (obtener el espectro del frame), necesitamos un proceso para la detección de picos espectrales, y otro para estimar la frecuencia fundamental del frame a partir de éstos. Para el cálculo de la energía, utilizaremos un objeto de la clase `Energy`, disponible también en CLAM, la cual realiza el cálculo de la energía a partir de un espectro que le es pasado como parámetro.

```
Energy energy; // instancia
Tdata e = energy(spec); // cálculo de la energía
```

El fragmento de código anterior, es un ejemplo de lo sencillo que resulta el cálculo de la energía a partir de su espectro mediante un objeto de la clase `Energy`. En nuestra implementación no lo haremos exactamente así, ya que la energía la guardaremos en un objeto de la clase `SegmentDescriptors`, la cual albergará en su interior un atributo de la clase `FrameDescriptors` que a su vez mantendrá un atributo de la clase `SpectralDescriptors` encargada de guardar el valor de energía del frame. Esto es igualmente sencillo y el siguiente fragmento de código así lo demuestra.

```
Energy energy;
tmpFrameD.GetSpectrumD().SetEnergy(energy(spec.GetMagBuffer()));
mSegmentDescriptors.GetFramesD().AddElem(tmpFrameD);
```

Aquí, `tmpFrameD` es una instancia de la clase `FrameDescriptors`, `spec` es el espectro del cual calcularemos su energía, y `mSegmentDescriptors` es una instancia de la clase `SegmentDescriptors`. El mostrar fragmentos de código como los anteriores no será una práctica habitual en esta memoria, ya que de hacerse así supondría tener que presentar aquí grandes cantidades de código, lo cual no aportaría gran cosa y dificultaría el seguimiento de los conceptos al introducir una cierta cantidad de “ruido”; de todas formas, en este caso se ha creído oportuno hacerlo con la finalidad de mostrar lo sencillas que pueden ser las cosas mediante el uso de CLAM, una vez los objetos han sido configurados de forma correcta. Recordar, no obstante, que la totalidad del código fuente de la aplicación se encuentra disponible en el CD que acompaña al proyecto para aquél que desee consultarlo. A continuación se presenta un diagrama estático de clases para ilustrar los elementos principales en el proceso de análisis.

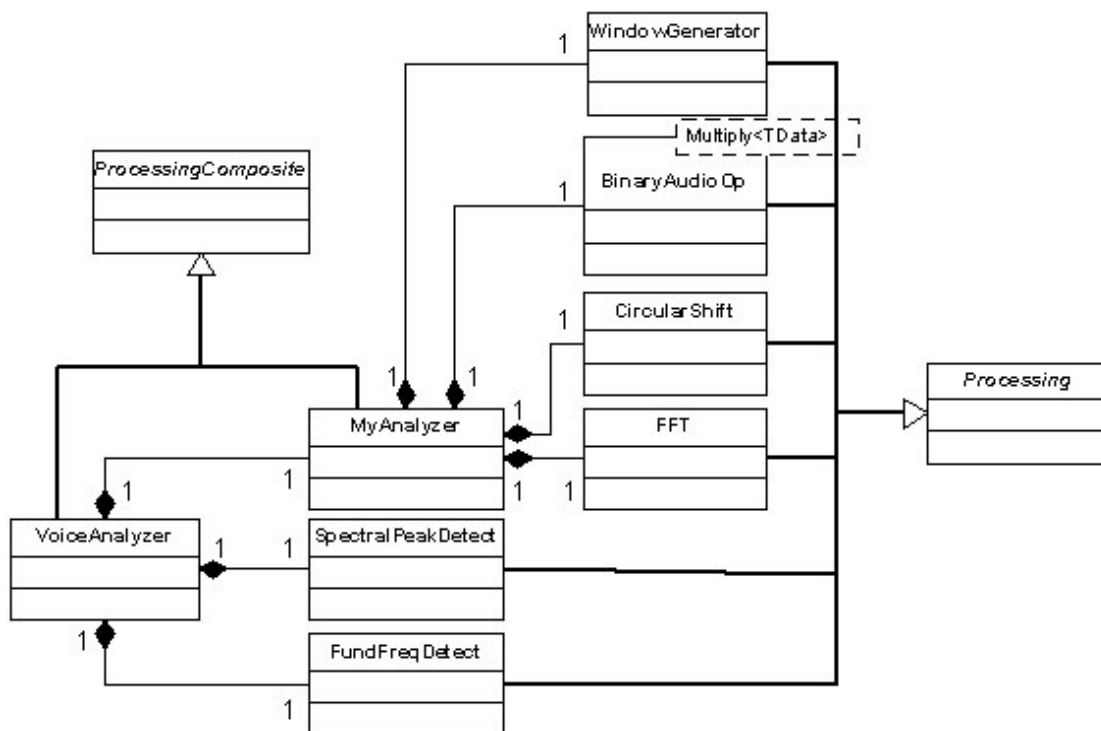


Fig 23. Diagrama simplificado de clases para el análisis.

En la figura 23 se observa que la clase `VoiceAnalyzer` tiene como miembros más relevantes una instancia de la clase `MyAnalyzer`, utilizada para obtener el espectro, una instancia de la clase `SpectralPeakDetect`, usada para el cálculo de los picos espectrales, y una instancia de la clase `FundFreqDetect`, utilizada para el cálculo de la frecuencia fundamental a partir del conjunto de picos espectrales devueltos por el proceso de detección de picos. La instancia de la clase `Energy` comentada anteriormente se hace de forma transparente, ya que se realiza en el interior de un método `Do` de la clase `VoiceAnalyzer`. Toda la información extraída en el análisis se mantiene en estructuras derivadas de `ProcessingData`. En este caso, el método `Do` recibe como parámetro una instancia a un objeto de la clase `Frame`, en donde se guarda parte de la información (frame de audio original, espectro, f_0 del frame...), y los valores de energía se almacenan en un objeto de la clase `SegmentDescriptors`, tal como se ha indicado con anterioridad.

Nótese la presencia del miembro de la clase `BinaryAudioOp`; ésta es una clase paramétrica (template), utilizada para realizar el producto del frame de audio de entrada con la envolvente determinada por el proceso de enventanado; en este caso se hará uso de una ventana tipo Hamming. La clase utilizada para el producto de la ventana por el frame de audio de entrada es, en realidad, una declaración `typedef` hecha a partir de la clase paramétrica `BinaryAudioOp`, y denominada `AudioMultiplier`. Tal como se puede ver en el diagrama, la operación que le es pasada como parámetro a la clase `BinaryAudioOp`, es precisamente la multiplicación. El proceso `CircularShift` garantiza la condición de fase cero durante el análisis, y consiste en hacer un intercambio (swap) de la parte izquierda del buffer con la parte derecha, tomando como referencia la posición central.

Con el fin de clarificar el orden de las acciones que se llevan a cabo durante el proceso de análisis, se presenta un diagrama que lo refleja con un poco más de detalle que en el diagrama de la figura 22.

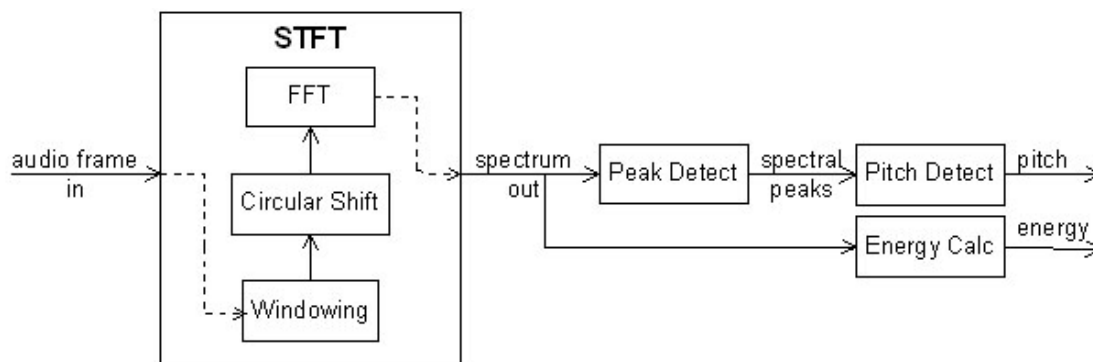


Fig 24. Diagrama de bloques extendido para el proceso de análisis.

Aunque no figure en el diagrama, se aplica un factor de zero-padding al frame de audio de entrada (ver 2.2.5). Como ya se ha comentado, el zero-padding aumenta el número de bins por Hz, favoreciendo así una mayor precisión en el cálculo de los picos espectrales. El tamaño de hop size empleado es igual a $(wSize-1)/2$, siendo $wSize = 513$ el tamaño de la ventana.

A partir de los datos calculados en el análisis, tenemos la información requerida para la extracción de la melodía contenida en la señal de audio original. Para realizar este proceso, asignaremos responsabilidades a la clase `MelodyAnalyzer`, la cual incorporará los mecanismos necesarios para llevar a cabo la tarea.

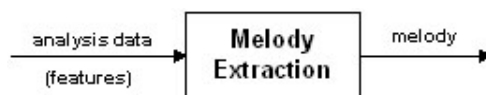


Fig 25. Extracción de melodía.

La clase `MelodyAnalyzer` realiza la operación de extracción de melodía por medio del método público `AnalyzeMelody(...)`, que recibe como parámetro la información necesaria para llevar a cabo el proceso. En primer lugar, se aplica el proceso de segmentación mediante el uso de la clase `Segmentator`, disponible en CLAM. El segmentador es configurado para procesar en base a los datos de energía y frecuencia fundamental presentes en la señal, obtenidos en la fase de análisis; a partir de dichos datos, el proceso realizado por el objeto de la clase `Segmentator` devuelve información, incorporando una lista de segmentos hijo que contienen las fronteras correspondientes a los límites temporales estimados, que definen la segmentación del audio original; a continuación, se hace una aproximación media de la energía y frecuencia fundamental de cada nota a partir de las fronteras que determinan los tiempos inicial y final de la misma y los datos de energía y f_0 calculados en el proceso de análisis.

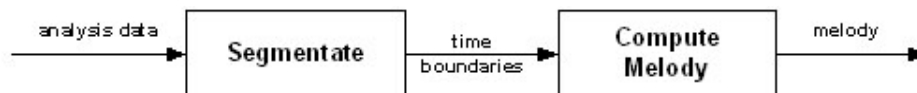


Fig 26. Segmentación y cálculo de melodía.

5.2.3 Soporte MIDI: conversión y salida

Recordemos en primer lugar, el conjunto de requisitos que debe cumplir la aplicación en referencia a las prestaciones MIDI.

- Conversión MIDI a partir de la melodía obtenida en el proceso de extracción de la misma.
- Almacenamiento del resultado de la conversión en un archivo MIDI.
- Reproducción de la melodía MIDI obtenida.

El proceso de conversión se realiza de una forma realmente sencilla. Tenemos una melodía disponible, fruto del proceso de extracción comentado en el apartado anterior. Entonces a partir de los valores contenidos en dicha melodía, podemos obtener otra melodía equivalente en valores MIDI.

Como se ha indicado anteriormente, los valores obtenidos en el proceso de extracción de la melodía (frecuencia, energía ...) se mantienen en un objeto de la clase `Melody`. A partir de estos valores puede calcularse fácilmente un equivalente MIDI, y estos nuevos valores calculados pueden ser almacenados en un objeto de la clase `MIDIMelody`, también disponible en CLAM. La pregunta es: ¿quién debe hacerse cargo de realizar esta tarea? Una buena candidata para ello es la clase `MelodyAnalyzer` comentada en el apartado anterior, dado que precisamente es en el interior de esta clase donde se obtiene la melodía en el formato guardado en el objeto de la clase `Melody`; así pues, el mapeo a valores MIDI se hará paralelamente a la obtención de la melodía en su formato inicial.

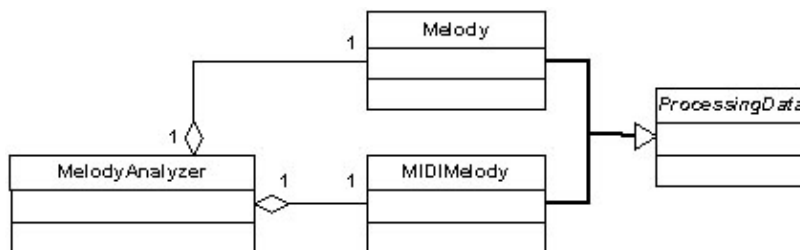


Fig 27. Clase MelodyAnalyzer.

En la clase `Melody`, se guarda una lista con todas las notas que forman la melodía, cuya información se encuentra en objetos de la clase `Note`. La clase `Note`, incorpora un método para el cálculo del valor MIDI de la nota a partir del valor de su frecuencia, y otro para obtener el valor de la velocidad de la nota a partir de su energía; entonces lo único que debemos hacer es

usar estos métodos para llenar una estructura de la clase `MIDIMelody`, con tal de obtener el equivalente en representación MIDI. De esta forma ya tenemos disponible una melodía MIDI, a partir de la cual podemos realizar el proceso de reproducción de la misma y el almacenamiento del archivo MIDI en disco; siendo los valores temporales de las notas, los mismos que se mantienen en el objeto de la clase `Melody`.



Fig. 28 Conversión MIDI.

Para la reproducción de la melodía MIDI diseñaremos e implementaremos una nueva clase, a la cual denominaremos `MIDIMelodyPlayer`. La reproducción MIDI, de igual modo en que se hace en el caso de la reproducción de audio, se realizará lanzando un hilo de ejecución (thread), con el fin de que el usuario pueda parar el proceso si así lo desea; por tanto, el diagrama de estados para este modo de reproducción es el mismo que el presentado para la reproducción de audio en la figura 20. Ya que el proceso de reproducción se inicia con la creación del objeto, los datos necesarios son pasados como parámetros al constructor de la clase. En este caso, los datos relevantes son la melodía MIDI, el identificador de dispositivo MIDI empleado y el programa (instrumento) con el cual se interpretará la melodía.

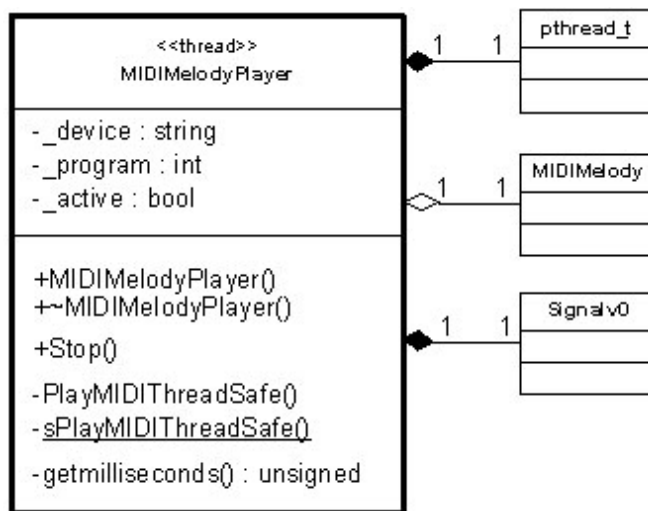


Fig. 29 Diagrama estático para la clase MIDIMelodyPlayer.

En el diagrama de diseño de la clase `MIDIMelodyPlayer`, se han incorporado todos los métodos y atributos, tanto públicos como privados. Esto se ha hecho así porque al tratarse de un diagrama con una única clase, hay espacio suficiente y la comprensión resulta rápida y sencilla. Además, la explicación que se hará a continuación, también contribuye a entender de forma completa el funcionamiento de las clases para la reproducción de audio, que han sido presentadas de manera muy simplificada, dado que la idea que subyace en la implementación del reproductor MIDI es muy similar a la que se aplica para la reproducción de audio.

En el constructor de la clase `MIDIMelodyPlayer`, se crea un hilo de ejecución cuyo código es el contenido en el método estático `sPlayMIDIThreadSafe`. En este método estático, se hace una llamada al método `PlayThreadSafe`, en el interior del cual se llevan a cabo todas las acciones necesarias para la reproducción. Igual que sucede con el audio, el MIDI también tiene un manejador encargado de gestionar la E/S. Esta responsabilidad recae sobre la clase `MIDIManager` de CLAM, y en nuestro caso controlaremos la reproducción haciendo uso de la clase de control `MIDIOutControl`, la cual se configura por medio de un objeto de la clase `MIDIIOConfig`. Los objetos `MIDIOutControl` son configurados y empleados para el envío de

mensajes MIDI. Para la reproducción de la melodía usaremos principalmente controles de volumen, programa (instrumento) y note on. El pseudocódigo para el método de reproducción `PlayThreadSafe` se muestra a continuación.

INICIO

```

- Inicializar variables
- Configurar controles MIDI (program, volume, note on)
- Iniciar manejador MIDI
- Asignar programa y volumen

active := true

mientras se tengan notas disponibles
    si no active → break
    - reproducir nota
fin mientras

enviar panic
si active → enviar señal indicando reproducción finalizada

```

FIN

Panic es un mensaje de control equivalente a enviar un note off a todas las notas (ver 2.3.3). El mensaje panic se emplea como solución cuando el usuario detiene la reproducción antes de que la nota en curso haya recibido su correspondiente mensaje note off. La señal de reproducción finalizada es enviada, en caso de ser necesario, por un objeto de la clase `Signalv0`, la cual forma parte del namespace `SigSlot` del framework. El método `getmilliseconds` se utiliza para controlar los inicios y finales de nota, y el método `Stop` detiene la reproducción. En el interior del método `Stop` únicamente se ejecuta una instrucción:

```
active = false;
```

Como se puede observar en el pseudocódigo, esta instrucción causa la salida del bucle de reproducción, produciéndose la finalización inmediata del proceso.

La escritura del archivo MIDI se ha realizado bajo la norma SMF (Standard MIDI File), que será comentada a continuación; por ahora diremos que se trata de un archivo en código ASCII cuya estructura es la siguiente:

```

MThd [length of header data]
[header data]
MTrk [length of track data]
[track data]
MTrk [length of track data]
[track data]
...

```

Para la escritura de archivos MIDI implementaremos una clase a la que llamaremos `MIDIFileWriter`, la cual ofrecerá un método a través de su interfaz pública, adecuado para la realización de la tarea.

```
void Write(const MIDIMelody& melody, int program, const std::string& filename);
```

Este método recibe como parámetros: la melodía MIDI con la serie de notas que formarán la pista MIDI del archivo, el entero correspondiente al programa (instrumento) con el cual se interpretará la melodía, y el nombre del archivo a escribir. En nuestro caso, se trata de un archivo que contiene una sola pista, por lo que su estructura estará compuesta por la cabecera seguida de los datos correspondientes a dicha pista.

La cabecera se sitúa al inicio y contiene información básica relativa al archivo (formato, número de pistas y división). El identificador de cabecera (`MThd`) y la longitud de la misma se codifican como palabras de 32 bits, y para representar la parte de datos se emplean palabras de 16 bits. La parte de datos puede tomar diversos valores, según las necesidades del archivo a escribir; aquí se mencionan los empleados para el caso que nos ocupa; para tener más

información acerca del formato de los archivos MIDI se pueden consultar las referencias web citadas en el capítulo 10 relativas a este tema.

Se utilizará el formato 0, que indica que el archivo contiene una única pista multicanal, el número de pistas es 1 como ya se ha indicado anteriormente, y la división, a la que asignaremos un valor de 96, indica la resolución en ‘ticks’ por cuarto de nota. Como no indicamos la signatura temporal ni el tempo de la pista, se le asignarán valores implícitos. Cuando no se define signatura temporal se le asigna el valor por defecto de 4/4, y en el caso del tempo, el valor es 120, expresado en beats por minuto. La cabecera de nuestro archivo tendrá el siguiente aspecto expresado en valores hexadecimales:

```
4D 54 68 64 MThd (palabra de 32 bits)
00 00 00 06 longitud de la cabecera en bytes (palabra de 32 bits)
00 00      formato 0 (palabra de 16 bits)
00 01      una pista (palabra de 16 bits)
00 60      96 ticks por cuarto de nota (palabra de 16 bits)
```

Como se puede deducir, para formar la cabecera necesitamos métodos adecuados para la escritura de palabras de 16 y de 32 bits; también se puede observar que las palabras de 32 bits no se tienen en cuenta para indicar el valor de longitud de la cabecera.

A continuación de la cabecera escribiremos la pista (track), donde estarán representados los mensajes MIDI que serán interpretados por algún reproductor MIDI o secuenciador. Una pista está compuesta de eventos, precedidos por un delta-time. El delta-time representa la cantidad de tiempo que transcurre entre el final de un evento y el inicio del evento siguiente.

Una pista puede contener distintos tipos de eventos:

```
[event] = [MIDI event] | [sysex event] | [meta-event]
```

En nuestro caso utilizaremos solamente eventos MIDI (*MIDI event*). Tenemos una pista en la que usaremos un único canal al cual asignaremos un determinado instrumento (programa), y un valor de volumen. El resto de eventos serán mensajes note on, ya que para el caso del note off usaremos un mensaje note on con valor de velocidad igual a cero.

La melodía MIDI que deseamos reproducir se encuentra almacenada en un objeto de la clase *MIDIMelody*, donde los valores de inicio y fin de nota están expresados en segundos en coma flotante, por lo que implementaremos un método que convierta los segundos a ticks con el fin de escribir el delta-time correspondiente para cada evento. Además, los valores de delta-time se deben representar como valores de longitud variable (en bytes) en el interior del archivo; entendiendo por longitud la cantidad de bytes necesarios para representar el valor; por tanto, también se debe disponer de un método para escribir el delta-time de cada evento como un valor expresado en longitud variable.

La escritura de la pista comienza con el identificador de pista (*MTrk*), seguido de la longitud en bytes de los datos que la forman. Estos valores son expresados en palabras de 32 bits, y al igual que ocurre en el caso de la cabecera, la cuenta real de bytes comienza a continuación. Al inicio del track todavía no conocemos su longitud, por lo que al término de la escritura de la pista, deberemos regresar a la posición adecuada en el archivo para escribir la longitud correcta. Siguiendo estas directrices, el pseudocódigo para el método *Write* puede ser el siguiente:

INICIO

- Inicializar variables.
- Abrir archivo para escritura.
- Escribir cabecera.
- Escribir pista.
- Cerrar archivo.

FIN

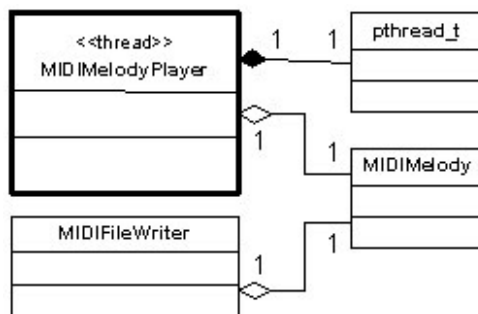


Fig. 30 Diagrama de clases simplificado para el soporte MIDI.

5.3 Visualización

Los diferentes procesos que se aplican sobre el audio de entrada al sistema, tales como el análisis o la extracción de melodía, generan información diversa acerca de características relativas a dicha entrada de audio. Es conveniente mostrar toda esta información generada por el sistema, con el fin de ofrecer una perspectiva al usuario acerca de los resultados obtenidos; una forma de producir esta realimentación la obtenemos a través del sistema de reproducción mediante el sentido auditivo, pero interesa disponer además de una percepción visual que la complemente. Llegado este punto es inevitable comenzar también la introducción a algunos de los elementos que formarán parte de la interfaz de usuario.

Los requisitos de visualización para el sistema son:

- Vista de forma de onda de la señal de audio original.
- Vista de análisis comprendiendo audio, energía y frecuencia fundamental.
- Vista editable del resultado de la segmentación (melodía).
- Vista editable tipo pianola (melodía MIDI).

Como funcionalidades adicionales a las anteriores tenemos:

- Consulta de valores en la vista de análisis.
- Sincronización entre los diferentes tipos de vista.
- Visualización dinámica durante la reproducción de audio.

Como solución a estos problemas implementaremos widgets que resolverán grupos de requisitos relacionados.

V2MAnalysisDisplay: vista de audio original, análisis y vista dinámica en la reproducción.

V2MSegmentationEditor: vista editable del resultado de la segmentación.

V2MPianoRoll: vista editable tipo pianola.

Un elemento esencialmente necesario en todos los casos, es una superficie adecuada sobre la cual generar los gráficos representativos de la información mostrada; para ello diseñaremos un lienzo habilitado para mostrar gráficos bidimensionales a partir de la clase `QWidget` disponible en el toolkit gráfico Qt, el cual ya ha sido comentado en un capítulo anterior. Llamaremos a esta clase `V2MDataDisplay`, la cual derivará de `QWidget` tal como se ha comentado. Al derivar de `QWidget`, disponemos de métodos virtuales, donde pueden usarse funciones del API OpenGL, los cuales pueden ser sobrescritos con tal de adecuarlos a nuestras necesidades.

En primer lugar, crearemos una clase denominada `V2MDataDisplay`, que utilizaremos para la visualización de datos de audio, energía y frecuencia fundamental; además, esta superficie deberá responder al evento producido por la pulsación del botón izquierdo del ratón sobre algún punto perteneciente al área de dibujo. La coordenada relativa al punto de pulsación, será empleada para trazar una línea vertical que indique el punto seleccionado, y a partir de dicho punto se mostrará al usuario información sobre valores de amplitud, energía, f_0 y tiempo instantáneo, mediante un panel provisto de etiquetas para esta finalidad. Una vez implementada

la superficie de generación, podemos hacer las instancias que deseemos con tal de mostrar una imagen comparativa de los datos presentados; en el caso de este widget, hemos dicho que se utilizará para mostrar datos de audio original, energía y f_0 ; entonces podemos tener una lista con tres superficies, las cuales pueden ser mostradas en torre con el fin de ofrecer una visión general de todos los datos. La siguiente figura muestra esta idea de forma esquemática.

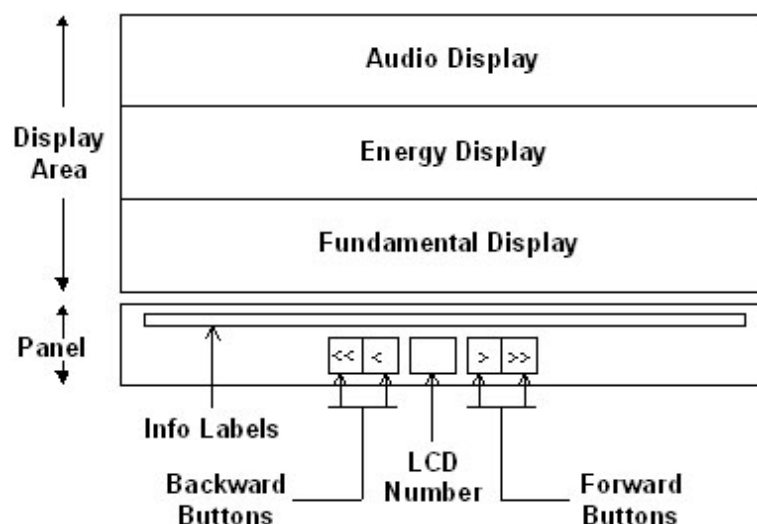


Fig. 31 Diseño para el widget V2MAnalysisDisplay.

Como se puede ver en la figura, tenemos tres superficies de generación, las cuales se muestran al observador como una sola; y cuando aparece la línea vertical al pulsar con el ratón sobre alguno de los lienzos, se puede propagar el valor de la coordenada x al resto de superficies con tal de que dicha línea sea trazada en las tres superficies de forma que parezca una sola línea a la vista del usuario. En el panel inferior, se colocarán etiquetas que mostrarán la siguiente información:

- Amplitud de la señal en el instante t .
- Energía de la señal en el instante t .
- Frecuencia fundamental de la señal en el instante t .
- Instante t .
- Duración total de la señal.

El instante t es el tiempo relativo a la coordenada x seleccionada mediante la pulsación del ratón sobre el área de display.

La vista de análisis ofrece dos modos: Full y Step by Step (o frame to frame). En el modo Full se muestra la señal completa, y en el modo Step by Step el resultado del análisis se presenta en frames consecutivos de 1 segundo de duración; es por este motivo que se ha colocado un grupo de botones de avance y retroceso, mediante el uso de los cuales puede navegarse a lo largo de la señal, mostrándose en todo momento el número de frame actual, en el display LCD que se encuentra entre el grupo botones de avance y retroceso. En el caso del modo Full no necesitamos navegar por la señal, ya que ésta se muestra de forma completa, por lo que, para este modo el grupo de botones estará desactivado. Como se puede intuir, para mostrar la forma de onda del audio original se utilizará el display superior y se ocultarán los otros dos (energía y fundamental), y en este caso el grupo de botones de navegación también estará desactivado. Además de esto, se debe tener en cuenta que los modos Full y Step by Step estén sincronizados en referencia al valor temporal seleccionado mediante la pulsación del ratón sobre el área de display.

La clase V2MAnalysisDisplay derivará de QWidget, y para ubicar los elementos del panel inferior usaremos la clase contenedor QFrame; a todo esto comentar que cuando el nombre de alguna de las clases mencionadas comience por 'Q' se entiende que dicha clase

forma parte del toolkit Qt. Las superficies de generación se colocan en torre en el orden indicado en la figura 31, y las etiquetas informativas, los botones y el display LCD se ubican en el interior del panel definido por el objeto de la clase `QFrame`, de tal forma que los elementos queden distribuidos en el widget tal como se muestra en la figura 31. Una vez se tienen todos los elementos que componen el widget, debemos alimentarlo con la información que debe mostrar; en este caso deberemos proporcionarle los datos de audio original, y los datos de energía y frecuencia fundamental obtenidos en el proceso de análisis.

El widget `V2MAnalysisDisplay`, también implementa la visualización dinámica de la forma de onda durante la reproducción de audio. Para lograr esto, debemos pensar en un mecanismo por el cual enviar los datos de audio según se van reproduciendo; como ya sabemos, en el proceso de reproducción se envía la señal a trozos hacia el dispositivo de audio para que éste los vaya reproduciendo, ya que no podemos enviar toda la señal de golpe; entonces sería cuestión de ir enviando los datos de cada trozo al display, de forma que éste se fuera actualizando, y de esta manera mostrar los datos de forma dinámica durante el proceso de reproducción. El toolkit Qt dispone de un sistema de señales y ranuras realmente útil del cual haremos uso sin lugar a dudas; pero para este caso, y dado que la información la envían las clases de reproducción de audio presentadas en el apartado 2.2, las cuales forman parte del sistema, debemos pensar en una alternativa con el fin de no producir acoplamiento entre el sistema y la interfaz de usuario; para ello emplearemos el sistema de señales y ranuras del namespace `SigSlot` incluido en el framework CLAM, e ideado para dar soporte al Visualization Module (CLAMVM). Por tanto, habrá una señal que envíe los datos del frame de audio que está siendo reproducido, y una ranura asociada a un método que recibirá dichos datos como parámetro con tal de manipularlos de forma adecuada. En el caso de la reproducción de audio original los datos enviados serán los del audio, y en el caso de la reproducción de frecuencia fundamental o frecuencia fundamental después de la segmentación (melodía), los datos serán los generados por un oscilador.

La clase `V2MDataDisplay` empleada como superficie de generación para el widget implementado por la clase `V2MAnalysisDisplay`, nos servirá como base para la creación de otra superficie de generación, a la cual denominaremos `V2MDataEditDisplay`, y que utilizaremos en la implementación del widget `V2MSegmentationEditor`. Para mostrar los resultados de la segmentación, lo haremos de manera que puedan ser comparados con los datos originales (audio, energía y f_0), además, las líneas verticales estarán presentes en todo momento para indicar los límites temporales de las notas resultantes del proceso de segmentación.

La clase `V2MDataDisplay` ofrece una superficie sobre la cual generar los datos de audio, energía y f_0 , así como la posibilidad de trazar líneas verticales; todo esto puede ser aprovechado por la clase `V2MDataEditDisplay`, en la que se extenderá la funcionalidad con tal de que pueda mostrar también los segmentos horizontales correspondientes a la duración de cada nota, en relación al valor de energía o frecuencia fundamental que le ha sido asignado. Entonces, la superficie de generación implementada en la clase `V2MDataEditDisplay` podrá mostrar, o bien los datos que muestra la clase `V2MDataDisplay`, o bien los datos generados en el proceso de segmentación según sea conveniente; además, la forma de tratar el evento de ratón que se produce para el caso del botón izquierdo, debe ser modificado con tal de hacer posible la edición. En la siguiente figura se muestra el esquema para la disposición de los elementos que componen el widget que será implementado por la clase `V2MSegmentationEditor`.

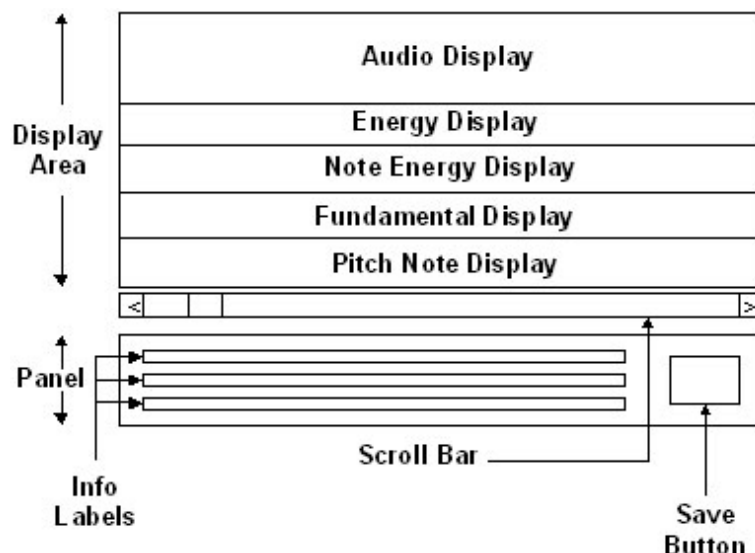


Fig. 32 Diseño para el widget V2MSegmentationEditor.

En la figura, se puede observar que el widget está provisto de 5 superficies de generación; en las cuales se muestra, tanto la información de análisis como la de segmentación, con el fin de hacer una comparativa entre ambos tipos de dato. En estas vistas, también se muestran los límites temporales (líneas verticales) que determinan el inicio y fin de cada nota de la melodía obtenida a través del proceso de segmentación, permitiendo además la edición de dichos límites por parte del usuario. La edición permitirá al usuario la modificación de los límites temporales de las notas que forman la melodía, con el fin de que éste tenga la ocasión de hacer un ajuste fino de la misma; además de esta funcionalidad, también se permitirá la eliminación de notas. La edición se realizará por medio del ratón; si el usuario pulsa con el botón izquierdo sobre alguna de las líneas verticales correspondientes a algún límite temporal de alguna nota, entonces el cursor del ratón cambiará de aspecto para indicar al usuario que dicho límite puede ser modificado, arrastrando el ratón a derecha o a izquierda mientras se mantiene el botón pulsado; cuando el usuario deje de pulsar el botón izquierdo del ratón, la línea vertical correspondiente al límite temporal que está siendo editado quedará en la posición deseada, y la longitud de los segmentos horizontales relativa al espacio temporal empleado por cada nota, se actualizará según el valor determinado por la nueva posición. Si se desea eliminar alguna nota de la melodía, no se tiene más que juntar la línea vertical que define el inicio de la nota con la que define su fin, para que la nota desaparezca, dejando así de formar parte de la melodía; también se pueden encadenar notas haciendo coincidir el fin de una nota con el inicio de otra.

En el panel inferior se muestra información relativa a los datos de audio original y análisis, en comparación con los obtenidos en la segmentación para la nota que está siendo editada; además, se tiene la opción de guardar los cambios mediante la pulsación del botón *Save*. Aparte del botón *Save*, también se dispondrá de un menú emergente, el cual aparecerá como respuesta a la pulsación con el botón derecho del ratón sobre el área de display; en este menú, además de la opción *Save*, también estarán disponibles las opciones *Undo* y *Discard*, que darán la opción al usuario de deshacer el último cambio, o descartar todos los cambios realizados a partir de la última acción de tipo *Save*; respecto de la opción de *Undo*, el sistema permitirá al usuario deshacer hasta un total de 20 acciones. Para poder deshacer una acción se debe tener información sobre el estado del sistema antes de realizarla; por lo que para guardar los estados por los que pasa la melodía durante su edición utilizaremos una pila de estados; entonces los sucesivos estados se van apilando en dicha pila, y si se desea deshacer alguna acción lo único que hay que hacer es desapilar el estado anterior.

El panel inferior dispone de tres filas de etiquetas, a través de las cuales se informará al usuario acerca de los resultados obtenidos en el análisis y en la segmentación para las notas que componen la melodía extraída a partir del audio de entrada.

- Primera fila: amplitud, energía y frecuencia fundamental en un instante t .
- Segunda fila: energía y frecuencia fundamental cuantificada para la nota seleccionada en esos momentos.
- Tercera fila: tiempo de inicio, tiempo de finalización, y duración total de la nota que está siendo editada actualmente.

La *barra de scroll* tendrá la misión de permitir el avance y retroceso en el tiempo a lo largo de la señal, ya que resultaría inadecuado mostrar toda la señal completa, dado que ésta puede resultar demasiado extensa, lo cual dificultaría su correcta visualización.

Para la distribución de los elementos, utilizaremos un objeto de la clase `QFrame`, que contendrá las cinco superficies de generación (instancias de la clase `V2MDataEditDisplay`), dispuestas en torre tal como se muestra en la figura 32; para definir el panel inferior, se procederá de forma similar a como se hizo en el caso del widget implementado por la clase `V2MAnalysisDisplay`, haciendo uso de otro objeto de la clase `QFrame`. El widget `V2MSegmentationEditor` será alimentado con los datos resultantes de los procesos de análisis y de segmentación, es decir, audio original, energía, f_0 y melodía.

El aspecto del widget `V2MPianoRoll`, que ofrece una vista tipo pianola, nos lo muestra el esquema de la siguiente figura.

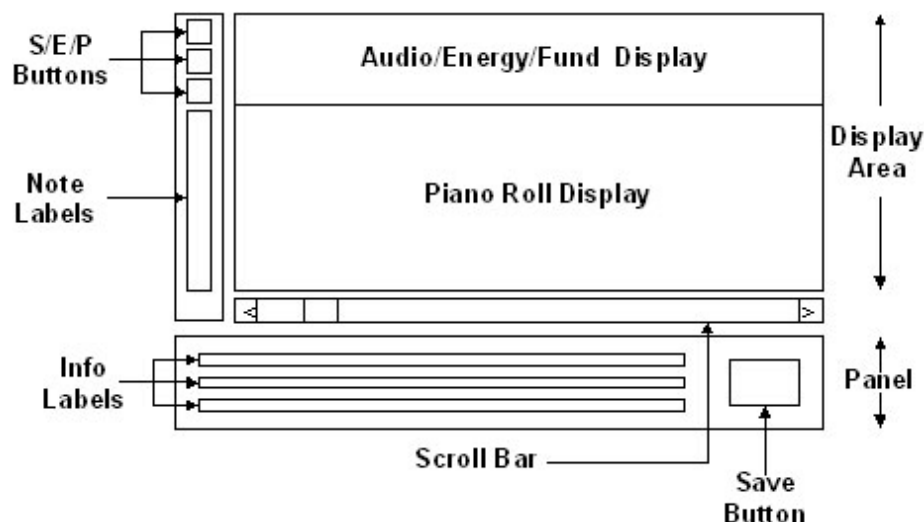


Fig. 33 Diseño para el widget `V2MPianoRoll`.

Para el display superior usaremos instancias de la clase `V2MDataEditDisplay`, derivada de `V2MDataDisplay`, y el tipo de información mostrada (audio, energía y f_0) estará controlado por los botones situados a la izquierda: [S]ignal (audio), [E]nergy (energía), [P]itch (f_0). Las etiquetas que se encuentran bajo los botones en la parte izquierda sirven para indicar las doce notas de la escala cromática: C, Db, D, Eb, E, F..., y cada una corresponde a una de las doce filas en que está dividida la rejilla de la pianola. El display tipo piano roll está compuesto por una rejilla $12 \times N$, donde N es el número de columnas, siendo dicho número proporcional a la duración de la señal de entrada, ya que cada columna tendrá una duración de 1 segundo.

Las notas se distribuyen en la rejilla de forma que queden en la posición correcta respecto de su valor temporal y altura (pitch). Para representar la octava a la cual pertenece cada nota utilizaremos un código de colores; esto es posible porque nunca se da el caso de que existan notas que se reproduzcan de forma simultánea; ya que éstas se reproducen siempre de forma secuencial; por tanto, no hay ninguna razón aparente por la cual se deba colocar otra barra de scroll para expandir la rejilla en vertical además de en horizontal. El rango de octavas será (2..6); esto no cubre todas las octavas posibles en el General MIDI, pero supera con creces la tesitura de voz de cualquier cantante.



Fig. 34 Código de colores para la representación de las octavas.

En la figura 34 podemos ver el color asignado a cada una de las octavas en el rango de 2 a 6. En todos los casos, la intensidad del color será proporcional al valor de velocidad de la nota que se está representando; a menor velocidad, la tonalidad de la nota será más oscura, y a mayor velocidad la nota se mostrará con una tonalidad más clara.

Para la implementación de la superficie de generación de la vista tipo pianola crearemos una clase denominada `V2MPREditDisplay`, la cual derivará de `V2MDataEditDisplay`. De esta forma, podemos aprovechar las funciones de edición que ofrece la clase base, y lo único que debemos hacer es sobrescribir el método de dibujo con tal de mostrar la información en los rectángulos temporales correspondientes a cada nota con una rejilla 12 x N de fondo. Además de sobrescribir el método `paintGL()`, también deberemos incorporar métodos que nos permitan obtener el color adecuado para la nota en cada caso; pero toda la funcionalidad de edición es heredada de la clase `V2MDataEditDisplay`.

Durante el diseño de los widgets que permitirán los distintos tipos de vista para el sistema de visualización de la aplicación, nos hemos podido dar cuenta de cómo el concepto de herencia incorporado en el lenguaje C++ nos ha facilitado la tarea a lo largo de todo el proceso.

Las etiquetas del panel inferior del widget que implementa la clase `V2MPianoRoll`, muestran la misma información que en el caso del widget `V2MSegmentationEditor`, salvo en el caso de los valores MIDI, los cuales sustituyen a los de energía y pitch cuantificado que se mostraban en el caso de la segmentación. La nueva información es la siguiente:

- Key MIDI: valor MIDI entre 0 y 127 correspondiente a la altura (pitch) de la nota.
- Velocidad: valor de velocidad (fuerza o intensidad de la nota) entre 0 y 127.
- Pitch: expresado en nomenclatura universal y acompañado de la octava a la cual pertenece la nota (ej Ab3 – la bemol de la tercera octava).

El mecanismo empleado para deshacer acciones es análogo al utilizado en el caso de la edición en la vista de segmentación, es decir, apilar y desapilar estados.

Para sincronizar las vistas de segmentación y pianola utilizaremos el sistema de señales y ranuras que incorpora Qt; esto no produce acoplamiento entre la interfaz y las clases pertenecientes al dominio, ya que las señales se envían y reciben por elementos GUI. Cada vez que se produzca alguna acción de edición en uno de los widgets, este enviará una señal al otro, el cual al recibirla realizará la misma acción. Una idea alternativa a esta es la solución propuesta por el patrón de diseño Observer, que implementa una solución elegante; pero ya que en nuestro caso únicamente debemos sincronizar dos elementos, hacer esto a través del sistema de señales y ranuras de Qt supone una solución más sencilla e igualmente válida. En el caso de que el número de observadores suscritos a un notificador fuera mayor o incluso indeterminado, la solución propuesta por el patrón Observer no sería sólo la más elegante, sino también la más sencilla y eficaz. Únicamente existe un problema que se daría en ambos casos: si un widget envía al otro una señal de notificación indicando que ha realizado una acción de edición, el receptor de la señal al realizar la acción de edición correspondiente, también envía una señal al otro widget indicando que está realizando una acción de edición, hecho que provoca un bucle infinito de envíos y recepciones; por tanto, hay que encontrar una solución para que esto no se produzca. La solución parece simple: el emisor solamente enviará la señal en caso de que su widget se encuentre visible en esos momentos, y el receptor solamente hará caso de la señal si en esos momentos el widget receptor no se encuentra visible.

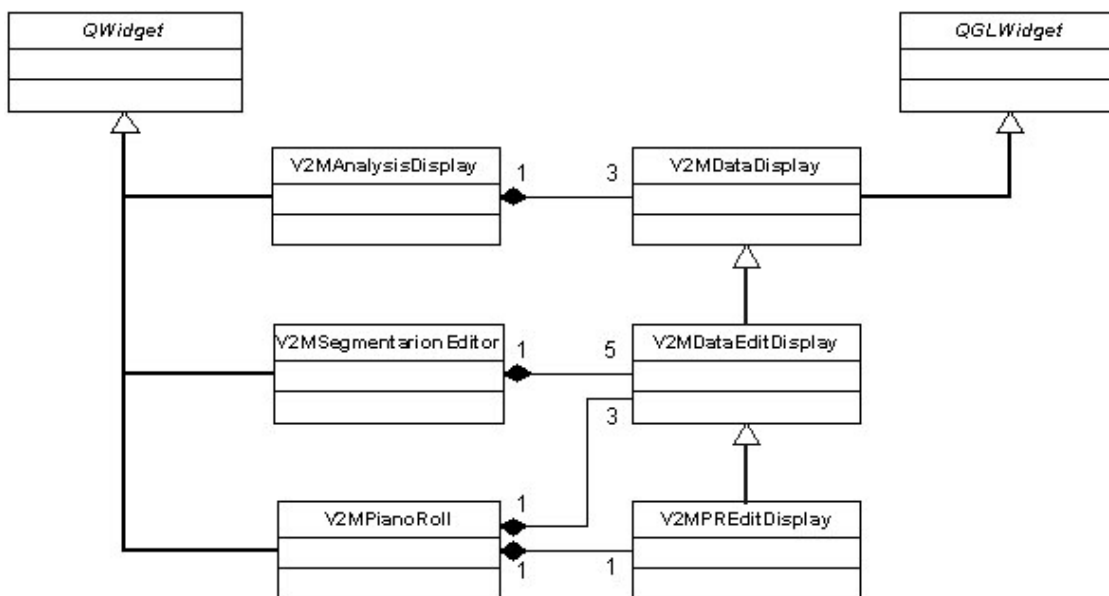


Fig. 35 Diagrama de clases simplificado para el sistema de visualización.

En este apartado dedicado al sistema de visualización y edición que incorpora la aplicación, se ha presentado una descripción global de la idea de diseño e implementación empleados, y se recuerda que para tener información detallada al respecto se dispone del código fuente.

5.4 XML

Es posible que los datos generados por la aplicación, en referencia a la melodía extraída a través de los procesos de análisis, se desee almacenar en algún tipo de archivo, de forma que se encuentre disponible para su uso en entornos de aplicación diferentes. Un formato muy extendido dada su sencillez y potencia es XML.

XML es un formato basado en texto utilizado para representar datos en forma jerárquica; para la representación de los datos, utiliza etiquetas de inicio y fin para cada uno de los elementos de la jerarquía, colocando el valor del elemento entre su par de etiquetas correspondiente. Como se trata de una organización jerárquica, un elemento puede contener a su vez otros elementos. En nuestro caso nos interesa tener la posibilidad de almacenar, una representación de los datos correspondientes a la melodía y su equivalente MIDI generadas por el programa.

CLAM ofrece soporte XML para el almacenamiento y carga de componentes, por lo que para guardar la melodía o la melodía MIDI, únicamente nos hace falta hacer uso de la clase XMLStorage. Podemos recuperar un componente desde un archivo XML mediante el método `Restore(comp, "filename.xml")`, siendo el componente a cargar el objeto pasado como primer parámetro y el nombre del archivo con los datos el segundo parámetro del método. Lo que nos interesa en este caso es almacenar la información, por lo que el método que debemos utilizar es `Dump(comp, "myComp", filename)`, siendo el primer parámetro el componente cuya información deseamos almacenar, el segundo parámetro el nombre del componente (primer tag de la jerarquía), y el nombre del archivo a guardar el tercer parámetro del método.

La responsabilidad de almacenar la información contenida de la melodía y la melodía MIDI extraídas, debe ser asignada a alguna clase. Según el Experto en Información, deberíamos asignar esta responsabilidad a la clase que tenga la información necesaria para realizar la tarea; la clase que mantiene esta información es `MelodyAnalyzer`.

Entonces, para almacenar la melodía MIDI en un archivo XML, nos basta con hacerlo a través del siguiente método:

```
void MelodyAnalyzer::StoreMIDIMelody(const std::string& filename)
{
    XMLStorage x;
    x.UseIndentation(true);
    x.Dump(midiMelody, "Analyzed_MIDIMelody", filename);
}
```

El método `UseIndentation` se utiliza para aplicar tabulaciones en el texto, con el fin de clarificar el orden en la jerarquía.

5.5 El sistema de ayuda

Toda aplicación software debe ofrecer al usuario algún tipo de soporte, a través del cual favorecer la comprensión en relación a los aspectos de funcionamiento del sistema. Dado el carácter multiplataforma del Sistema de Conversión de Voz a MIDI, lo más indicado es incorporar un sistema de ayuda en HTML, teniendo en cuenta además, que Qt incluye herramientas que facilitan la integración de un browser como parte del sistema.

Para el sistema de ayuda de la aplicación, implementaremos un sencillo navegador para visualizar documentos HTML a través de la interfaz de usuario; para ello, crearemos en primer lugar el conjunto de documentos HTML que formarán el sistema de ayuda, y a continuación diseñaremos un widget que nos permita visualizar los documentos, y navegar a través de los enlaces que los relacionan.

La clase perteneciente al toolkit Qt que utilizaremos en la implementación del widget es `QTextBrowser`, y el visualizador incorporará, además de la funcionalidad de navegación propia de HTML, la posibilidad de ir a la página anterior o a la página siguiente, en caso de que estén disponibles, y la oportunidad de volver a la página de inicio (home) en cualquier momento. Lo único que debemos hacer es indicarle al objeto de la clase `QTextBrowser` la ruta (path) al directorio en el cual se encuentran los archivos que forman el sistema de ayuda, y la página de inicio, en este caso `index.htm`; seguidamente conectaremos los botones que gestionarán las acciones comentadas anteriormente, a las señales adecuadas emitidas por el objeto que implementa el browser, con el fin de conseguir la respuesta esperada.

La ventana de visualización de la ayuda del sistema se creará dinámicamente a petición del usuario, y se destruirá cuando éste decida cerrarla.

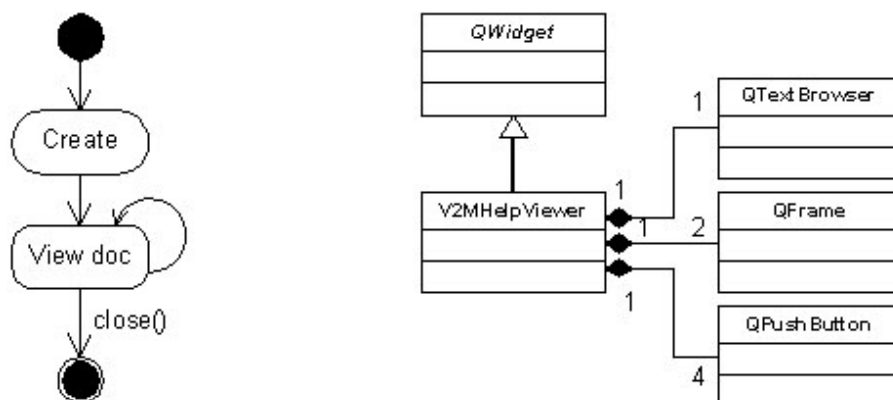


Fig. 36 Diagrama de estados y diseño para el sistema de ayuda.

En el diagrama de estados de la figura 36 podemos ver, que después de la creación del objeto que implementa el sistema de ayuda, pueden visualizarse los documentos, y que al dar la orden de cierre el objeto se destruye. El diagrama de clases nos indica los elementos que componen el widget que implementa la ventana de visualización de la ayuda.

5.6 Uniendo todas las piezas

En los apartados anteriores de este capítulo se han descrito las distintas partes de las que consta la aplicación, por lo que ahora es tiempo de pensar en la forma de comunicación entre los elementos que componen el sistema, con el fin de obtener los resultados esperados.

Como ya se ha comentado, el sistema permite al usuario realizar una serie de tareas, las cuales han sido presentadas anteriormente en el capítulo dedicado a los requisitos de la aplicación, y esquematizado mediante un diagrama de casos de uso. Estas tareas pueden ser vistas como actividades del sistema, por lo que es interesante disponer de alguna herramienta que nos ofrezca una visión de los flujos de trabajo y de los procesos o casos de uso.

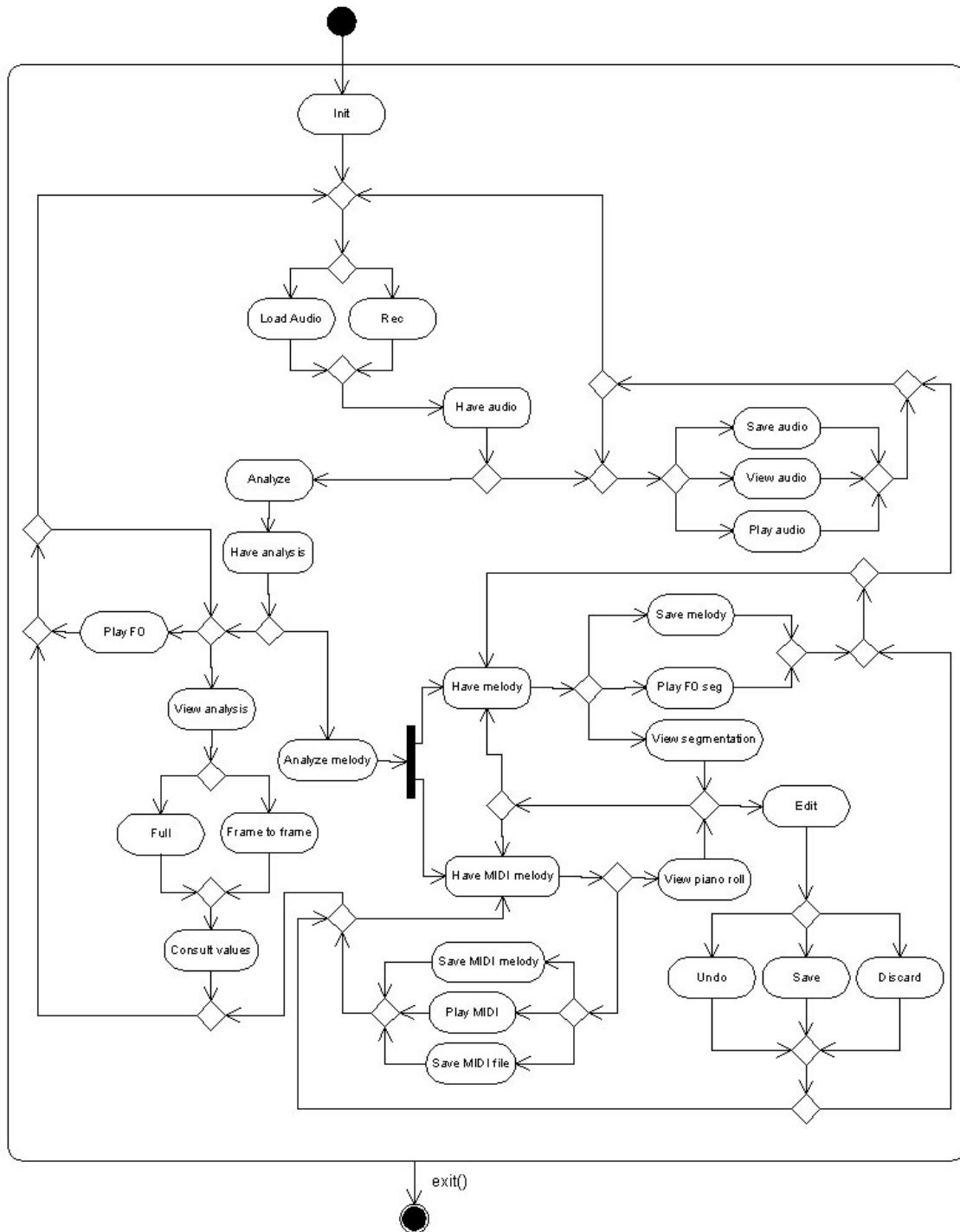


Fig. 37 Diagrama de actividades.

Por otra parte, las clases que pertenecen al dominio de la aplicación pueden ser agrupadas en módulos cohesivos. Para el Sistema de Conversión de Voz a MIDI podemos diferenciar los siguientes grupos:

- Entrada/Salida de audio.
- Análisis.
- Salida MIDI.

En una arquitectura por capas, los módulos mencionados podrían ser los elementos de la capa del dominio, en la cual se llevarían a cabo los procesos realizados en el núcleo del sistema.

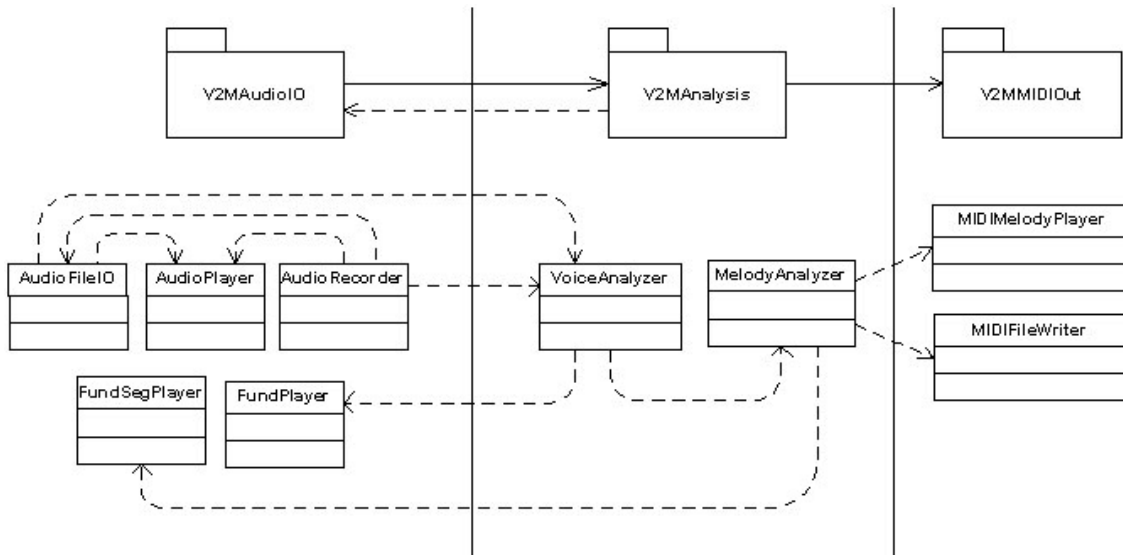


Fig. 38 Flujo de información en la capa del dominio.

En la figura 38 se muestran las clases pertenecientes al dominio de la aplicación; en realidad no están presentes todas las clases que componen el sistema completo, como por ejemplo la clase MyAnalyzer, ya que ésta es un miembro de la clase VoiceAnalyzer (ver figura 23). Se puede observar el flujo de información entre los elementos; por ejemplo, la clase AudioRecorder se encarga de abastecer a la clase VoiceAnalyzer proporcionándole el audio que debe ser analizado; por otra parte, la clase VoiceAnalyzer le sirve a la clase MelodyAnalyzer la información necesaria para la extracción de la melodía contenida en el audio original; una vez obtenida la melodía MIDI, la información puede ser solicitada por las clases MIDIFileWriter o MIDIMelodyPlayer para realizar su tarea.

Como ya sabemos, las funcionalidades del sistema serán ofrecidas al usuario a través de una interfaz gráfica. El diseño del interfaz se hará de forma que facilite, en la medida de lo posible, un uso intuitivo de la aplicación, lo cual en principio no debería presentar ningún obstáculo, dado que el número de funciones disponibles no es elevado. Se dispondrá de una barra de menú con los grupos y operaciones asociadas representados en la siguiente tabla:

File		Analysis	View		?
Audio	Load...	Analyze	Original audio		Help...
	Save...	Extract Melody	Analysis	Full	About...
MIDI	Save MIDI file...	Step by Step			
Melody	Save melody...		Segmentation		
	Save MIDI melody..		Piano Roll		

El interfaz de usuario incluirá una barra de estado en la cual se mostrarán mensajes informativos; por ejemplo, durante la reproducción de audio original podría mostrarse el mensaje: "Playing original audio...". También se colocarán botones para controlar los procesos

de registro y reproducción; para el caso de la reproducción se dispondrá de un grupo de botones de radio a través del cual indicar el modo de reproducción (Audio, MIDI...); asimismo, habrá un grupo de controles *combo box* para seleccionar las opciones disponibles para el caso del MIDI (dispositivo e instrumento). El área de visualización, estará ocupada por alguno de los widgets presentados en el apartado dedicado a la visualización. En la siguiente figura se muestra el esquema de diseño para la interfaz gráfica de usuario.

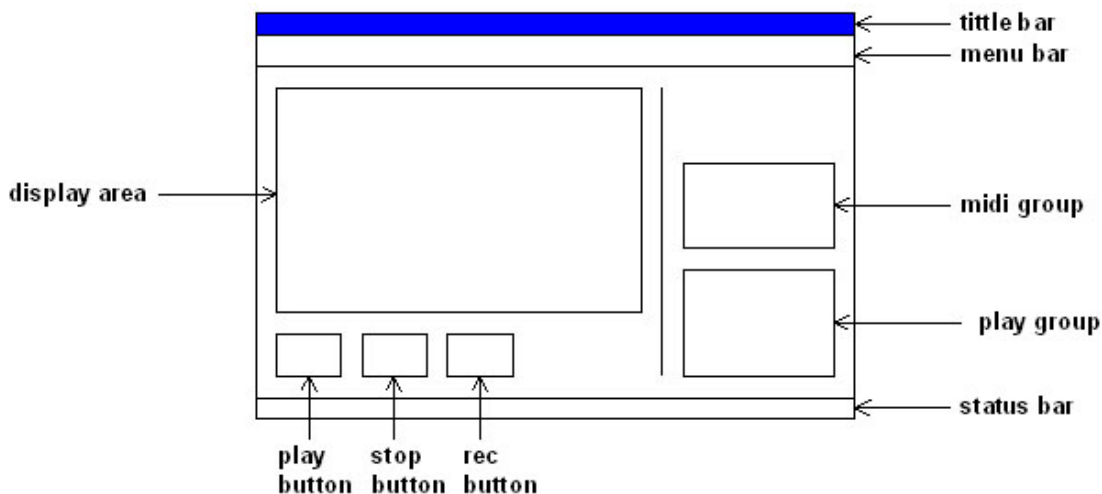


Fig. 39 Disposición de elementos en la ventana principal.

Para implementar la ventana principal del interfaz crearemos una nueva clase a la que llamaremos *V2MGUI*. El interfaz gráfico se encarga de ofrecer una vista de los datos, y de gestionar los eventos producidos por las acciones de usuario; por lo tanto, requiere servicios de los objetos del dominio, para obtener de ellos la información necesaria en cada caso.

El principio de Separación Modelo-Vista establece que los objetos del modelo (dominio) no deberían conocer directamente a los objetos de la vista (presentación), por lo que es aconsejable añadir un nivel de indirección, mediante el cual establecer la comunicación entre los elementos del interfaz de usuario, y los objetos del dominio.

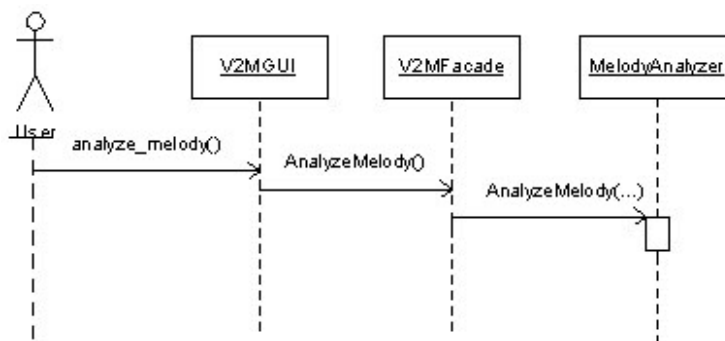


Fig. 40 DSS para el UC analizar melodía.

En la figura 40 se muestra un diagrama de secuencia del sistema (DSS), para la realización del caso de uso (Use Case) analizar melodía. En el diagrama podemos ver que el nivel de indirección mencionado anteriormente es incorporado a través de un controlador de fachada. El controlador de fachada es una clase que representa al sistema completo, y sirve de interfaz entre la vista y el modelo. En nuestro caso, es adecuado crear un controlador de fachada, debido a que la aplicación no maneja un excesivo número de eventos, por lo que dichos eventos pueden ser gestionados por una única clase controlador; en caso de tener que manejar un gran número de

procesos, habría sido preferible distribuir las responsabilidades entre varias clases controlador, preferiblemente implementando un controlador para cada caso de uso.

La clase `V2MFacade` será la encargada de mediar entre la interfaz gráfica y los objetos del dominio, captando los mensajes solicitados desde el interfaz, y redirigiendo las peticiones hacia los objetos del dominio encargados de realizar el trabajo; entonces, la clase `V2MFacade` incorporará métodos para delegar en los objetos del dominio, la realización de las tareas solicitadas desde el interfaz.

Un factor a tener en cuenta durante el diseño de un sistema software, es que las clases del dominio encapsulan la información, y el comportamiento relacionado con la lógica de la aplicación; las clases que implementan la interfaz gráfica son relativamente delgadas y son responsables de la entrada, salida, y captura de eventos del GUI, pero no mantienen datos ni proporcionan, de forma directa, ninguna funcionalidad de la aplicación. Algunas de las razones por las que hacer una separación entre el modelo y la vista son las siguientes:

- Dar soporte a definiciones de modelos cohesivos que se centren en los procesos del dominio, en lugar de preocuparse de las interfaces de usuario.
- Permitir separar el desarrollo de las capas del modelo y la interfaz de usuario.
- Minimizar el impacto de los cambios de los requisitos del interfaz sobre la capa del dominio.
- Permitir que se conecten fácilmente otras vistas a una capa de dominio existente.

...

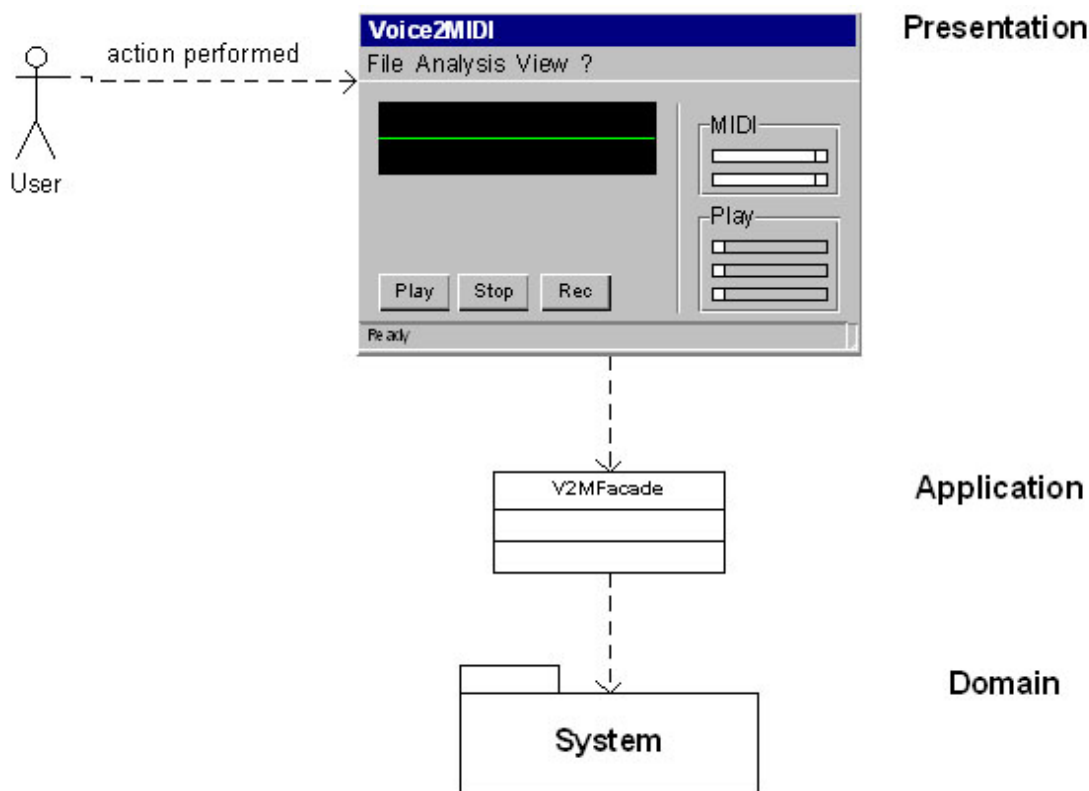


Fig. 41 Nivel de indirección entre la capa de presentación y la capa de dominio.

Cuando existe un solo controlador encargado de manejar todo el flujo de trabajo, como ocurre en el caso del controlador de fachada, la clase controlador suele formar parte de la capa de dominio, sirviendo como punto de acceso para la capa de presentación; por otra parte, cuando existe un mayor número de clases de control, como sucede con los controladores de casos de uso, es común crear otra capa que mantiene en su interior todas las clases control. De todas formas, nada nos impide situar el controlador de fachada como elemento único de la capa de aplicación tal como se muestra en la siguiente figura.

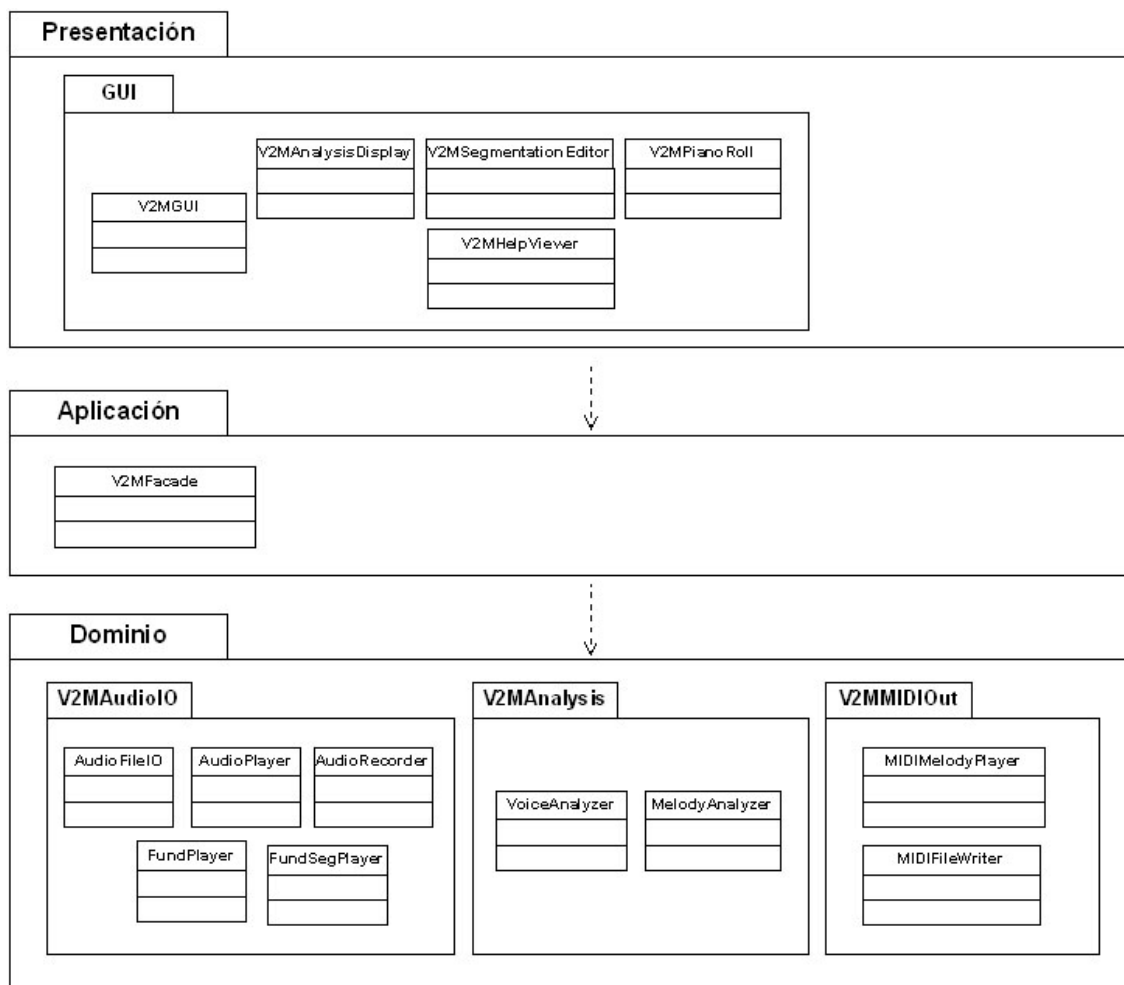


Fig. 42 Vista lógica del sistema mediante una arquitectura por capas.

Una arquitectura por capas, permite hacer una separación de los elementos de la aplicación, en áreas de responsabilidades distintas y relacionadas, con una separación clara y cohesiva de intereses. La colaboración y el acoplamiento es desde las capas más altas hacia las más bajas.

El patrón Capas se relaciona con la arquitectura lógica, describiendo la organización conceptual de los elementos del diseño independientemente de su despliegue físico, y definiendo un modelo general en N-niveles, donde cada nivel corresponde a una capa. El objetivo y número de capas varía de una aplicación a otra; para el diseño del Sistema de Conversión de Voz a MIDI se ha empleado una arquitectura en 3 niveles, donde cada nivel se corresponde con una capa y cada capa se organiza de la siguiente forma:

- **Presentación:** ventanas de la GUI.
- **Aplicación:** gestión de las peticiones de la capa de presentación; flujo de trabajo.
- **Dominio:** gestión de las peticiones de la capa de aplicación; servicios del dominio.

De forma general, el patrón Capas ofrece solución a los siguientes problemas:

- Los cambios en el código fuente se propagan a lo largo de todo el sistema.
- La lógica de la aplicación se entrelaza con la interfaz de usuario, entonces no se puede reutilizar con una interfaz diferente, ni distribuirse a otro nodo de proceso.
- Existe un alto acoplamiento entre distintas áreas de interés, por lo se hace difícil dividir el trabajo entre diferentes desarrolladores mediante límites claros.
- Debido al alto acoplamiento y a la mezcla de intereses, es difícil que la funcionalidad evolucione, que el sistema crezca de forma proporcionada, o que se actualice para utilizar nuevas tecnologías.

Después de haber descrito las distintas partes de las que consta el sistema, y presentado las clases que implementan la aplicación, estamos en condiciones de elaborar un diagrama estático de clases que ofrezca una visión global de todo el sistema.

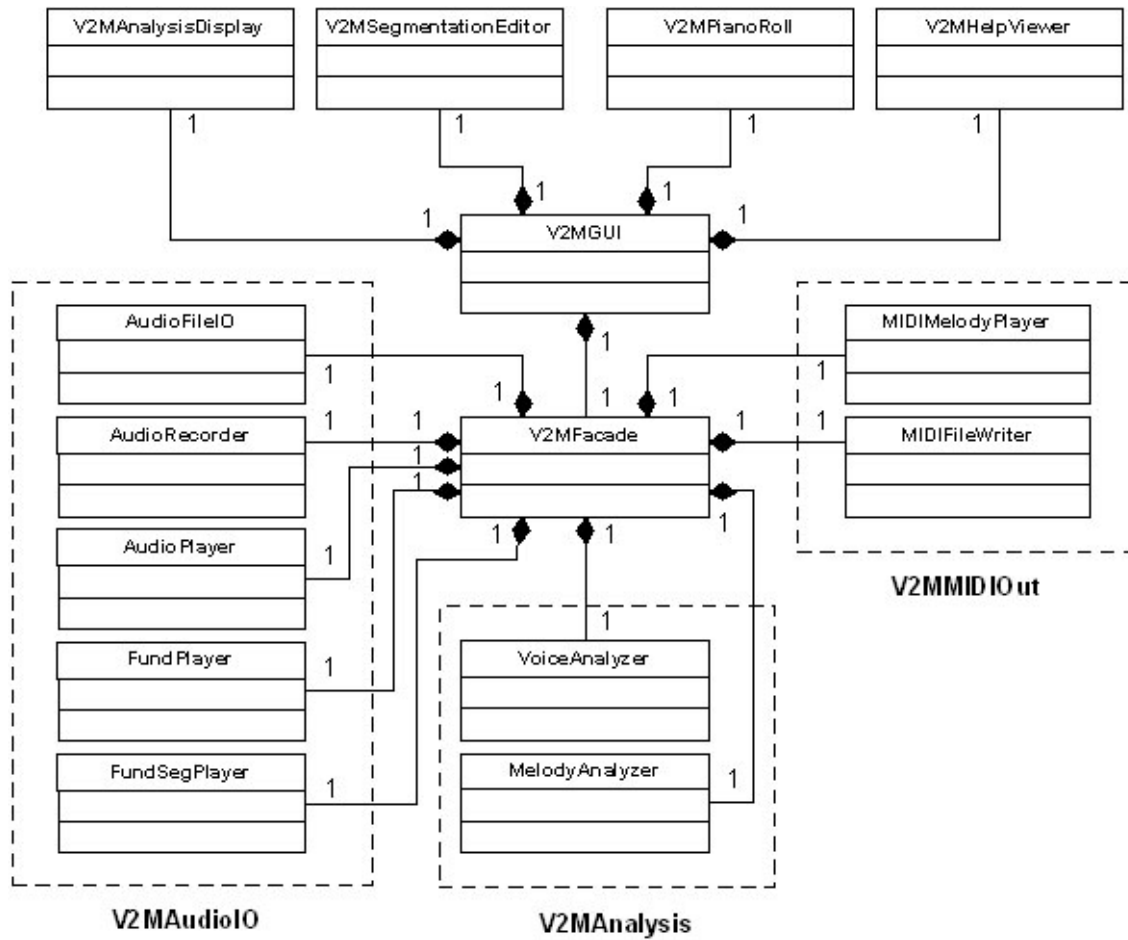


Fig. 43 Diagrama estático de clases para el Sistema de Conversión de Voz a MIDI.

En el diagrama de clases de la figura 43 no están presentes todas las clases utilizadas para implementar el sistema, como por ejemplo la clase *MyAnalyzer*, aunque no es estrictamente necesario incluir todas las clases en el diagrama de diseño; por otro lado, la clase *MyAnalyzer* no interactúa de forma directa con ningún objeto del dominio, ya que lo hace como atributo de la clase *VoiceAnalyzer*.

En el diagrama queda patente la separación de la capa de interfaz de la capa de dominio, ya que como se puede observar, no hay ninguna clase del dominio, de la cual exista conocimiento por parte de alguna clase que forme parte de la interfaz de usuario. La comunicación entre la capa de presentación y la capa de dominio se produce a través del controlador de fachada.

6. Conclusiones

Una vez desarrollada una primera versión del conversor MIDI, es hora de reflexionar un poco acerca del trabajo realizado y sobre lo que queda por hacer. En los apartados de este capítulo se realiza una pequeña valoración del prototipo implementado, mencionando en primer lugar los objetivos alcanzados respecto a la idea inicial; para continuar con un pequeño estudio de usabilidad, realizado a partir de la opinión de varios usuarios, que han tenido la amabilidad de ocupar parte de su tiempo, en realizar una serie de tareas y rellenar un breve cuestionario, el cual se encuentra en el anexo de este documento; seguidamente, se comparan los resultados en referencia a la calidad de la conversión MIDI, con los obtenidos por otras aplicaciones similares que están actualmente en el mercado; por último, se presentan algunas sugerencias sobre las líneas a seguir, en cuanto a futuras ampliaciones y mejoras.

6.1 Objetivos conseguidos y estado actual

Un proyecto fin de carrera situado en el contexto al que nos estamos refiriendo, tiene como finalidad, entre otras cosas, comprobar si el alumno ha adquirido cierta madurez en sus conocimientos, que le permita afrontar con éxito el desarrollo de una pieza de software que ofrezca alguna funcionalidad interesante. Aunque no puede decirse que el proyecto Voice2MIDI es de gran envergadura, sino más bien de pequeña escala, en él se tocan diversos temas esenciales del desarrollo de aplicaciones, como son: el acceso a dispositivos, el uso de frameworks, librerías de terceros, hilos de ejecución (threads), interfaces gráficas, etc. Hoy en día nadie quiere comenzar sus desarrollos desde cero, y es común apoyarse en entornos de trabajo, como en este caso CLAM, y en herramientas que faciliten la construcción de interfaces gráficas como Qt. No obstante, es posible que el desarrollador no conozca las herramientas elegidas para la implementación en un proyecto concreto; por este motivo, es importante adquirir cierta soltura en el uso de herramientas, en un espacio de tiempo relativamente corto.

En este proyecto se ha logrado alcanzar un buen número de los objetivos propuestos inicialmente, aunque deben refinarse ciertos aspectos con tal de mejorar el funcionamiento de la aplicación; en concreto, un objetivo no alcanzado de forma completa está directamente relacionado con la función principal del sistema: la conversión MIDI, ya que en este terreno queda mucho por hacer para obtener unos resultados respetables. Sin embargo, se ha logrado implementar el prototipo de una primera versión, cuyo funcionamiento parece estable y no presenta conflictos.

Se ha afrontado con cierto grado de éxito, la búsqueda de soluciones a problemas de diversa índole, que en el momento de la elección del proyecto eran desconocidos para mí a nivel práctico, dado que nunca había utilizado CLAM ni Qt, trabajado con hilos de ejecución ni diseñado la arquitectura de un sistema software completo, de las características del Sistema de Conversión de Voz a MIDI, y transformado dicho diseño en una implementación. Por tanto, pienso que el desarrollo de este proyecto ha contribuido de forma significativa para la comprensión de nuevas técnicas y conceptos importantes, así como servido como introducción a lo que representa un proyecto software en el mundo real.

En su estado actual, la aplicación incorpora todas las funcionalidades que han sido mencionadas en el capítulo anterior, dedicado al diseño e implementación.

6.2 Estudio de usabilidad

Es conveniente contar con la opinión de diversos usuarios acerca de las prestaciones ofrecidas por el producto, así como contrastar distintas opiniones respecto a la calidad, usabilidad, y demás factores que juegan un papel importante, para que la aplicación sea aceptada por el público. El que tiene la última palabra es siempre el usuario, el cual decidirá si utilizar o no el programa; si el usuario dice NO, entonces todos los recursos y esfuerzo empleados habrán sido en vano.

En este proyecto no se tiene la intención de comercializar el producto final, ya que en este caso, el desarrollo está motivado por el aprendizaje y por factores puramente pedagógicos; de todas formas, es igualmente interesante realizar un pequeño estudio de usabilidad, que por otra parte resulta de gran utilidad, entre otras cosas, para identificar anomalías en el funcionamiento, y pensar en la posibilidad de incorporación de nuevas funcionalidades echadas en falta por el usuario, o eliminar aquellas que no sean realmente necesarias.

El estudio de usabilidad se ha llevado a cabo de forma sencilla, por medio de un cuestionario de usabilidad. En primer lugar, se pide al usuario que realice una serie de tareas haciendo uso de la aplicación, y a continuación se le exponen una serie de enunciados en los cuales debe elegir una de las respuestas disponibles, y en caso de querer hacer algún comentario al respecto, puede añadirse en un espacio reservado para ello. El modelo de cuestionario empleado se encuentra en el anexo de la memoria.

El criterio empleado para hacer las puntuaciones respecto a la respuesta elegida por el usuario es el siguiente:

5. Muy Bien.
4. Bien.
3. Aceptable.
2. Mal.
1. Muy Mal.

Hay 12 enunciados cada uno de los cuales se puntúa en un rango de 1 a 5. El cuestionario ha sido rellenado por doce usuarios, por lo que el estudio ha sido realizado a partir de las opiniones de doce personas, todas ellas relacionadas de alguna manera con alguna de las áreas de la ingeniería. Para presentar los resultados de la encuesta, en primer lugar mostraremos una tabla con el porcentaje relativo al nivel de puntuación para cada uno de los enunciados. A continuación, se mencionarán los comentarios y sugerencias de los usuarios sobre algunas de las cuestiones consultadas. Finalmente, se incluye un gráfico de barras, que representa el resultado global para cada una de las doce cuestiones planteadas en el cuestionario, y se hace una valoración a nivel general.

		Muy Bien	Bien	Aceptable	Mal	Muy Mal
1	Facilidad de uso	33,33%	50,00%	16,67%	0,00%	0,00%
2	Sistema de reproducción	8,33%	50,00%	41,67%	0,00%	0,00%
3	Calidad en la conversión	33,33%	66,67%	0,00%	0,00%	0,00%
4	Información ofrecida	25,00%	58,33%	16,67%	0,00%	0,00%
5	Sistema de visualización	8,33%	33,33%	58,34%	0,00%	0,00%
6	Sistema de edición	8,33%	25,00%	50,00%	16,67%	0,00%
7	Representación de octavas	16,67%	58,33%	16,67%	8,33%	0,00%
8	Uso intuitivo	25,00%	33,33%	25,00%	16,67%	0,00%
9	Documentación de ayuda	58,34%	8,33%	33,33%	0,00%	0,00%
10	Instrumentos MIDI disponibles	8,33%	50,00%	41,67%	0,00%	0,00%
11	Interfaz gráfica	8,33%	50,00%	25,00%	16,67%	0,00%
12	Opinión global	33,33%	50,00%	16,67%	0,00%	0,00%

Fig. 44 Porcentajes de puntuación para los enunciados del cuestionario de usabilidad.

En la figura 44 se puede ver que las respuestas de los usuarios, tienden hacia la categoría que otorga una puntuación de 4 en el rango de 1 a 5; por otra parte, entre los usuarios que han rellenado el cuestionario, no ha habido ninguno que crea que alguno de los aspectos consultados pueda formar parte de la peor categoría, ya que nadie ha marcado la menor puntuación en ninguna de las preguntas. También se observa, que algunas personas creen que el sistema de edición, y algunos de los aspectos relacionados con la interfaz de usuario, no son demasiado acertados.

Los usuarios han centrado sus sugerencias en el sistema de reproducción y en la interfaz de usuario. Se echa de menos un indicador que muestre la progresión durante la reproducción, se opina que el sistema de edición es deficiente y restrictivo, y que la interfaz de usuario podría mejorarse para que fuera más atractiva, con el fin de captar la atención del usuario.

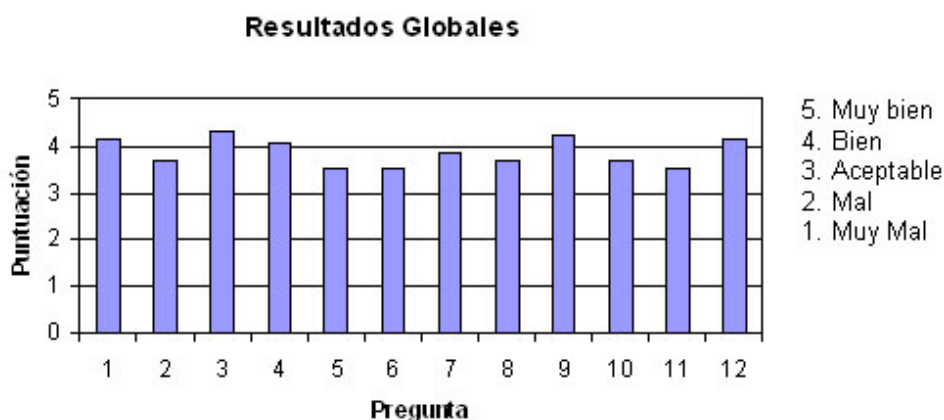


Fig. 45 Promedio en los resultados de la encuesta.

La gráfica de la figura 45 indica que la opinión de los usuarios acerca de la aplicación es buena, ya que el promedio se sitúa en torno a 4 para todos los enunciados planteados. Aunque es claro que se han de mejorar muchas cosas, y en el sondeo no han entrado en juego demasiados usuarios (sólo doce), la respuesta no ha sido mala, lo cual indica que el sistema puede ser útil, hecho que supone una satisfacción para el desarrollador, y anima a la continuidad en el trabajo para lograr mejores resultados y prestaciones.

6.3 Comparativa con aplicaciones similares

Actualmente existen algunas aplicaciones en el mercado, que realizan la conversión a MIDI a partir de una señal de audio de entrada. Estas aplicaciones, ofrecen un mayor número de opciones que el sistema desarrollado en este proyecto, y una interfaz gráfica más elaborada. No obstante, una característica interesante ofrecida por el Sistema de Conversión de Voz a MIDI y no incorporada en las aplicaciones elegidas en esta comparativa, es la posibilidad de editar los resultados obtenidos desde la propia aplicación, y disponer de vistas donde comprobar y consultar valores relativos al análisis de la señal. De todas formas, en esta comparativa se tiene en cuenta únicamente la característica considerada más importante: la calidad de los resultados obtenidos en la conversión.

Se han tomado cuatro aplicaciones, con tal de contrastar resultados, dos de las cuales (Akoff Composer y Widi) están incluidas en la categoría 'state-of-the-art' según un experimento que puede ser consultado en: <http://ismir2002.ismir.net/proceedings/02-FP04-3.pdf>

Aplicaciones utilizadas en la comparativa

Akoff Composer. Programa shareware desarrollado por Andrei Kovalev, para la conversión de señales musicales monofónicas a formato MIDI.

<http://www.akoff.com>

e-Xpressor. Programa de conversión de voz a MIDI.

<http://www.e-xpressor.com>

IntelliScore. Programa de conversión MIDI de Innovative Music Systems.

<http://www.intelliscore.net>

Widi. Programa de reconocimiento musical desarrollado por estudiantes rusos.

<http://www.widisoft.com>

Se ha realizado la conversión a MIDI de un archivo que contiene una señal de voz cantada tarareando; el archivo de audio es de corta duración, dado que la demo de uno de los sistemas (IntelliScore) impone una restricción, que únicamente permite la conversión de los 15 primeros segundos de la señal de entrada. El mismo archivo ha sido utilizado en todas las aplicaciones, incluido el sistema implementado en este proyecto.

Akoff Composer presenta mejores resultados, y además también funciona cantando con lyrics (letra de la canción), aunque en este último caso, la calidad en los resultados no es tan buena.

El resto de aplicaciones parece tener una calidad inferior respecto a la ofrecida por el sistema desarrollado en este proyecto. Por otra parte, Widi incorpora un buen detector de pitch, pero el problema que muestra es la inclusión de un mayor número de notas respecto de las que realmente forman parte de la melodía contenida en el audio de entrada.

Se observa que en todos los casos, la falta de precisión o el error tienden a producirse durante el proceso de segmentación, y se concluye que la dificultad principal se encuentra en esta etapa del proceso de conversión, por tanto, el esfuerzo debe centrarse principalmente en la búsqueda de técnicas y algoritmos que ofrezcan mejores resultados en la segmentación de la señal de audio original.

6.4 Desarrollo futuro

El contexto en el que se encuentra situado el conversor MIDI sugiere un gran número de ideas, las cuales pueden ser aplicadas en la mejora y ampliación del sistema. Las sugerencias para un desarrollo futuro que se comentan a continuación, indican los aspectos que deberían tener una mayor prioridad.

La mejora más necesaria, o más bien dicho: esencial es el refinamiento en el proceso de segmentación y extracción de melodía. En estos momentos, el sistema ofrece resultados relativamente coherentes cuando el audio de entrada no incorpora lyrics, y el usuario debe tararear ta,da... Una característica importante, que daría prestigio a la aplicación, pasa por el estudio de nuevas técnicas y algoritmos que permitan resultados aceptables en todos los casos, lo cual no es tarea fácil. Una vez conseguido este objetivo, lo realmente interesante sería que el sistema funcionara también en tiempo real; por lo que se tendrían disponibles dos modos de funcionamiento: tiempo real y fuera de tiempo.

A continuación se presentan otras sugerencias no tan esenciales, aunque no por ello dejan de tener interés.

- Permitir la integración de algoritmos de segmentación de terceros. De esta forma, el usuario podría utilizar la aplicación como entorno donde evaluar su propio algoritmo de conversión, sin necesidad de tener que programar los detalles de E/S de audio y MIDI, ni los aspectos relacionados con el sistema de visualización y la interfaz de usuario.
- Tener un mayor control sobre el sistema de reproducción. Podría añadirse un indicador de progreso sincronizado con la reproducción, como indicación de posición en cada momento. Además, sería una buena idea añadir una función 'Pause', que permitiera detener la reproducción de forma momentánea, y posteriormente continuarla a partir del punto de pausa. También debería ser posible seleccionar el punto en el que se desea iniciar el proceso de reproducción en cualquier caso.
- Posibilidad de almacenamiento y carga de archivos de proyecto específicos de la aplicación. Dado que el sistema incorpora algunas funciones de edición, sería interesante que la aplicación ofreciera soporte para la gestión de archivos de proyecto; se tendría la posibilidad de guardar archivos de proyecto, los cuales podrían ser cargados nuevamente por el usuario para hacer los ajustes que se crean oportunos.

Por otro lado, es evidente que las ideas están acotadas únicamente por el límite que marca la imaginación, por lo que muchas otras características imaginables podrían ser también añadidas.

Anexo

En este anexo se ha incorporado diverso material relacionado con el proyecto. El primer apartado contiene la planificación del proyecto, a continuación se han colocado los escenarios para los casos de uso del sistema; seguidamente se encuentra el cuestionario empleado en el estudio de usabilidad; y para finalizar se indican los pasos a seguir para construir la aplicación en las plataformas Windows y Linux, con los compiladores VC++ y gcc respectivamente, a través del entorno de gestión de proyectos incorporado en CLAM.

A.1 Planificación

La planificación de este proyecto está condicionada por el aprendizaje. Cabe destacar, que en la fecha de elección del proyecto, lo único que yo conocía sobre CLAM y Qt era su existencia, y en relación a las técnicas de ingeniería de software y diseño orientado a objetos, mis conocimientos eran igualmente escasos. De modo que tomé la decisión de matricularme como créditos de libre elección, en las asignaturas de IAM (Informática del Audio y de la Música), donde tomé un primer contacto con el framework CLAM y las técnicas de análisis espectral, e IS1 (Ingeniería de Software I), que me sirvió para conocer los conceptos básicos sobre ingeniería de software y diseño OO, y para practicar con el lenguaje C++.

A continuación de esta estrategia previa, podemos considerar que la planificación y desarrollo serio del proyecto tienen su inicio el día 8 de Enero de 2004, así que se dispone de aproximadamente 5 meses (si la entrega se realiza en Junio de 2004), para concluir el proyecto, incluyendo la implementación de la aplicación, memoria, y demás material relacionado.

En la gestión del proyecto se ha seguido un proceso iterativo, en el cual en cada iteración se han añadido nuevas funcionalidades, y refinado las ya implementadas en la iteración anterior en caso de que fuera necesario. Pero ¿qué se hace primero? ¿qué se hace en la siguiente iteración? La respuesta a qué es lo que hay que atacar en principio son las funcionalidades de mayor riesgo, con el fin de tener una primera versión del núcleo de la aplicación que trabaje de forma aceptable, para posteriormente ir adaptando las nuevas funcionalidades, que se han considerado de menor riesgo, sobre este esqueleto principal.

En el capítulo dedicado a los requisitos, ya fueron presentadas las funcionalidades del sistema, las cuales se enumeran ahora a modo de recordatorio, acompañadas del enunciado del problema a resolver.

Enunciado del problema: Se trata de implementar un sistema de conversión de voz cantada a MIDI, utilizando para ello técnicas de análisis espectral. El sistema, incorporará los mecanismos necesarios para permitir el almacenamiento y carga de archivos de audio, así como la capacidad de reproducir audio y MIDI. También deberá ser posible la visualización y edición de los datos obtenidos en el análisis, y el almacenamiento del resultado de la conversión en un archivo MIDI.

Características:

- Captura de señal de audio mediante micrófono.
- Almacenamiento y carga de archivos de audio en formato wav.
- Extracción de características de la señal mediante análisis espectral.
- Extracción de la melodía contenida en la señal y conversión a MIDI.
- Visualización y edición de resultados mediante distintos tipos de vista: señal original, análisis, segmentación y piano roll.
- Reproducción de audio y MIDI.
- Almacenamiento del resultado en un archivo MIDI.

A grandes rasgos, esta enumeración constituye los problemas que han de ser resueltos por el sistema, con el fin de ofrecer un servicio útil para el usuario. Se considera que los casos de mayor riesgo y complejidad, lo constituyen la realización del análisis de la señal para obtener las características necesarias (frecuencia fundamental y energía), y la segmentación de la

misma, que permita la obtención de la melodía contenida en dicha señal. De modo que los requisitos de prioridad más alta se encuentran en el analizador y el segmentador.

También sería conveniente poder tener una vista de los resultados del análisis, y reproducir el tono fundamental obtenido, así como la señal original, con el fin de asegurar la coherencia de dichos resultados respecto del audio que está siendo analizado. De forma que en la primera iteración deberán desarrollarse las primeras versiones del analizador, el segmentador, la reproducción de audio, y parte de la interfaz de usuario que incorpore las vistas adecuadas para los datos manejados por el programa. En este sentido, se puede pensar en una planificación en amplitud y superficial, que defina un bloque principal que será refinado y nuevamente ampliado en las iteraciones posteriores, por tanto, aquí podrían añadirse las funcionalidades de captura y almacenamiento de la señal, lo cual completaría, junto con el reproductor, una primera versión del motor de entrada/salida de audio de la aplicación.

En cada iteración, es bueno planificar únicamente los requisitos para la iteración actual, sin tratar de prever lo que pasará en las iteraciones siguientes, y ese es el criterio empleado durante el desarrollo de este proyecto. No obstante, en este capítulo de la memoria se incluye la planificación de todas las iteraciones, dado que la estructura de este documento así lo requiere.

En base a lo expuesto, podemos dividir el desarrollo del proyecto en cuatro grandes iteraciones, en cada una de las cuales se realizarán una serie de tareas, con el fin de implementar una solución conveniente en relación a los problemas planteados por los requisitos correspondientes a la iteración en curso. Dado que oficialmente el proyecto comienza paralelamente al inicio del curso, consideraremos ésta como la fecha de inicio. A continuación se presenta la planificación de forma esquemática; en cada iteración se muestra la fecha de inicio y fin y se presenta la lista de tareas a realizar.

Iteración 1. 29/09/03 – 05 /12/03

- Recopilación de información.
- Estudio de requisitos.
- Asimilación de conceptos fundamentales.
- Toma de contacto con el framework CLAM.

Iteración 2. 08/01/04 – 12/03/04

- Analizador y segmentador.
- Entrada/Salida de audio (captura, reproducción, almacenamiento y carga).
- La reproducción de audio incluye: audio original, frecuencia fundamental obtenida en el análisis y reproducción de la melodía extraída.
- Toma de contacto con el toolkit Qt.
- Visualización de la forma de onda de la señal original.
- Visualización frame a frame de los resultados obtenidos en el análisis, incluyendo: forma de onda de señal original, energía y frecuencia fundamental a lo largo del tiempo. Esta vista debe permitir al usuario la consulta de valores de amplitud, energía y frecuencia fundamental de forma interactiva, dado un instante de tiempo concreto.
- Visualización y edición de los resultados de la segmentación tipo melodía vs. resultados del análisis. La idea es una vista que presente la relación existente entre la melodía obtenida tras la segmentación y la señal original, su energía y f_0 en el tiempo. Esta vista permitirá al usuario la edición de los límites temporales de las notas que forman la melodía, así como la posibilidad de eliminar notas. La edición estará sincronizada con la reproducción de la melodía, y será posible deshacer acciones de edición, guardar cambios o descartar todos los cambios y volver a la situación inicial.
- Todas las funcionalidades anteriores serán presentadas al usuario a través de una interfaz gráfica provista de un menú que incorpore las acciones requeridas para la gestión de archivos de audio, análisis de la señal, extracción de la melodía, y las distintas vistas ya mencionadas anteriormente; botones para reproducción y captura de audio, y selectores para la elección del tipo de reproducción deseada (audio original, f_0 de análisis y f_0 de segmentación correspondiente a la melodía extraída).

Iteración 3. 15/03/04 – 28/05/04

- Revisión de los requisitos de la iteración 1.
- Visualización y edición tipo piano roll. La idea es una vista tipo pianola, que incorpore en su parte superior un área más reducida que muestre información sobre la señal original, su energía y frecuencia fundamental. Para este fin se colocarán controles que permitan la selección del tipo de información que se desee visualizar respecto de la señal original (audio, energía o f0). Al igual que en el caso de la vista y edición de la segmentación, se tendrá la posibilidad de deshacer acciones, guardar cambios y descartar todos los cambios realizados.
- Sincronización de la vista piano roll con la vista de segmentación, de forma que los cambios realizados en la vista de pianola se vean reflejados en la vista de segmentación, y viceversa.
- Reproducción MIDI.
- Almacenamiento MIDI.
- Completar la interfaz de usuario, y añadir controles para la gestión del MIDI (selección de dispositivo, instrumento, etc.).

Iteración 4. 01/06/04 – 18/06/04

- Revisión de los requisitos.
- Pruebas y validación.

En este capítulo se ha presentado una planificación un tanto informal, con el fin de ilustrar la estrategia empleada durante el desarrollo del proyecto. Los proyectos software se apoyan modelos estructurados, con el fin de organizar de forma coherente las actividades que se realizarán durante el desarrollo del producto. El **Ciclo de Vida** del software es un conjunto estructurado de actividades que tiene como finalidad el desarrollo de un sistema software. Existen varios modelos de ciclo de vida, cada uno de los cuales resulta adecuado dependiendo de la naturaleza del sistema que se desee implementar.

- **Desarrollo en Cascada:** Separa las fases de especificación y desarrollo.
- **Desarrollo Evolutivo:** Especificación y desarrollo están entrelazados.
- **Desarrollo Formal:** El sistema se construye a partir de un modelo matemático.
- **Desarrollo basado en la reutilización:** El sistema se configura a partir de componentes ya existentes.

De entre estos modelos genéricos de ciclo de vida, este proyecto se enmarca dentro del modelo de **Desarrollo Evolutivo**, ya que en un principio se partió de un pequeño conjunto de requisitos bien definido; pero durante el desarrollo, el sistema ha ido creciendo paralelamente a la implementación. Por tanto, los requerimientos del sistema se han entendido a medida que se ha ido desarrollando. En la fase inicial del desarrollo del Sistema de Conversión de Voz a MIDI, solamente se contemplaban algunos requisitos básicos:

- Registro de voz mediante micrófono.
- Análisis y segmentación.
- Conversión MIDI.

No se había hablado de aspectos como la reproducción de audio y MIDI, o el almacenamiento y carga de archivos. Aunque estaba claro que las funcionalidades del sistema, serían ofrecidas al usuario a través de una interfaz gráfica, no se había comentado nada sobre los tipos de visualización que ofrecería el sistema, y si en alguna de estas vistas se permitiría la edición o no de los resultados obtenidos a partir del análisis y la conversión MIDI. Todos estos requerimientos, han ido surgiendo durante el desarrollo del software. Por otro lado dadas sus características, este proyecto encaja perfectamente, dentro del conjunto de aplicaciones software donde suele emplearse el modelo de Desarrollo Evolutivo, ya que la finalidad es la implementación de un sistema de medida pequeña.

A.2 Escenarios de casos de uso

UC1

Caso de uso: Load Audio (cargar audio).

Contexto: El usuario quiere cargar un archivo de audio en formato wave.

Actores principales: Usuario.

Precondiciones: En el disco se encuentra disponible un archivo de audio en formato wave.

Postcondiciones:

- En el sistema hay audio disponible para ser reproducido.
- En el sistema hay audio disponible para ser analizado.
- En el sistema hay audio disponible para ser visualizado.

Escenario principal de éxito:

1. El usuario selecciona la opción de menu: File → Audio → Load...
2. El sistema muestra un cuadro de diálogo para cargar el archivo.
3. El usuario selecciona el archivo wave que desea cargar.
4. El usuario pulsa el botón de validación en el cuadro de diálogo.
5. El sistema carga el audio en memoria.
6. El sistema <<muestra la forma de onda del audio>> cargado.

Includes:

- View Waveform (ver forma de onda del audio original).

UC2

Caso de uso: Store Audio (almacenar audio).

Contexto: El usuario quiere almacenar en disco un archivo de audio en formato wave.

Actores principales: Usuario.

Precondiciones: Existe audio disponible para ser guardado en un archivo.

Postcondiciones: El archivo de audio queda almacenado en el disco.

Escenario principal de éxito:

1. El usuario selecciona la opción de menu: File → Audio → Save...
2. El sistema muestra un cuadro de diálogo para guardar el archivo.
3. El usuario selecciona la ubicación y escribe el nombre del archivo a guardar.
4. El usuario pulsa el botón de validación en el cuadro de diálogo.
5. El sistema almacena el audio en un archivo con el nombre indicado por el usuario.

UC3

Caso de uso: Rec (registrar audio).

Contexto: El usuario quiere registrar una señal de voz.

Actores principales: Usuario.

Precondiciones: El usuario dispone de un micrófono conectado a su computadora.

Postcondiciones:

- En el sistema hay audio disponible para ser reproducido.
- En el sistema hay audio disponible para ser analizado.
- En el sistema hay audio disponible para ser visualizado.
- En el sistema hay audio disponible para ser almacenado en disco.

Escenario principal de éxito:

1. El usuario pulsa el botón Rec.
2. El sistema muestra la vista de audio.
3. El sistema comienza el proceso de captura de audio.
4. El usuario tararea la melodía.
5. El usuario finaliza el tarareo de la melodía.
6. El usuario pulsa el botón **Stop**.
7. El sistema finaliza el proceso de captura de audio.

Extensiones:

- 3.1 El sistema muestra una vista dinámica de la forma de onda del audio durante el proceso de captura.

UC4

Caso de uso: Play Audio (reproducir audio original).

Contexto: El usuario quiere reproducir el audio original.

Actores principales: Usuario.

Precondiciones: Existe audio disponible para ser reproducido.

Postcondiciones: –

Escenario principal de éxito:

1. El usuario selecciona el botón de radio Audio en el grupo Play.
2. El usuario pulsa el botón **Play**.
3. El sistema comienza la reproducción del audio.

Extensiones:

- 3.1 Si la vista actual es la ofrecida por la opción de menú: View → Original audio, el sistema muestra al usuario una visualización dinámica de la forma de onda del audio durante el proceso de reproducción.
- 3.2 Si el usuario pulsa el botón **Stop**, el proceso de reproducción finaliza.

UC5

Caso de uso: View Waveform (ver forma de onda del audio original).

Contexto: El usuario quiere visualizar la forma de onda del audio original.

Actores principales: Usuario.

Precondiciones: En el sistema hay audio disponible.

Postcondiciones: –

Escenario principal de éxito:

1. El usuario selecciona la opción de menú: View → Original audio.
2. El sistema muestra una vista con la forma de onda del audio original.

UC6

Caso de uso: Analyze (analizar audio).

Contexto: El usuario quiere analizar el audio.

Actores principales: Usuario.

Precondiciones: El sistema tiene audio disponible para ser analizado.

Postcondiciones:

- El sistema se encuentra en condiciones para realizar la extracción de la melodía.
- Hay datos de análisis disponibles para ser visualizados y consultados.
- La opción de reproducción de frecuencia fundamental se encuentra disponible.

Escenario principal de éxito:

1. El usuario selecciona la opción de menú: Analysis → Analyze
2. El sistema comienza el proceso de análisis.
3. El sistema muestra un cuadro indicando el progreso en el análisis.
4. Cuando la barra de progreso alcanza el 100% el proceso termina.

UC7

Caso de uso: View analysis full (ver resultados completos del análisis).

Contexto: El usuario quiere visualizar los resultados del análisis de forma completa.

Actores principales: Usuario.

Precondiciones: El análisis ha sido realizado.

Postcondiciones: –

Escenario principal de éxito:

1. El usuario selecciona la opción de menú: View → Analysis → Full.
2. El sistema muestra una vista de los datos de audio, energía y f0 de la señal completa, mostrando también la duración total de la señal de audio en el panel inferior.

UC8

Caso de uso: Play Fund (reproducir frecuencia fundamental).

Contexto: El usuario quiere reproducir la frecuencia fundamental contenida en el audio.

Actores principales: Usuario.

Precondiciones: El análisis ha sido realizado, y se encuentran disponibles los datos requeridos para la reproducción de frecuencia fundamental.

Postcondiciones: –

Escenario principal de éxito:

1. El usuario selecciona el botón de radio Fundamental en el grupo Play.
2. El usuario pulsa el botón **Play**.
3. El sistema comienza la reproducción de la f_0 .

Extensiones:

- 3.1 Si la vista actual es la ofrecida por la opción de menú: View → Original audio, el sistema muestra al usuario una visualización dinámica de la sinusoide correspondiente al tono puro generado por un oscilador, durante el proceso de reproducción.
- 3.2 Si el usuario pulsa el botón **Stop**, el proceso de reproducción finaliza.

UC9

Caso de uso: View analisis frame to frame (ver resultados de análisis frame a frame).

Contexto: El usuario quiere visualizar los resultados del análisis frame a frame.

Actores principales: Usuario.

Precondiciones: El análisis ha sido realizado.

Postcondiciones: –

Escenario principal de éxito:

1. El usuario selecciona la opción de menú: View → Analysis → Step by Step.
2. El sistema muestra una vista de los datos de audio, energía y f_0 para un frame de 1 segundo de duración, incorporando el número de frame actual y la duración total de la señal.
3. El usuario pulsa algún botón de avance o retroceso para ver más datos de la señal.
4. El sistema actualiza la vista y el número de frame con los nuevos datos.
5. Se repiten 3 y 4 mientras el usuario lo desee.

UC10

Caso de uso: Consult values (consultar datos de análisis).

Contexto: El usuario quiere información de valores acerca del análisis realizado.

Actores principales: Usuario.

Precondiciones: El análisis ha sido realizado, y el tipo de vista actual mostrada por el sistema es alguna de las vistas de análisis.

Postcondiciones: –

Escenario principal de éxito:

1. El usuario pulsa con el botón izquierdo del ratón, sobre algún punto perteneciente al área de visualización de datos.
2. El sistema actualiza las etiquetas del panel inferior en la vista de análisis, con los valores de amplitud, energía, frecuencia fundamental y tiempo instantáneo, relativos a la posición seleccionada por el usuario mediante la pulsación del ratón.
3. Se repiten 1 y 2 mientras el usuario lo desee.

UC11

Caso de uso: Extract Melody (extracción de melodía).

Contexto: El usuario quiere extraer la melodía contenida en el audio original.

Actores principales: Usuario.

Precondiciones: El proceso de análisis ha sido realizado.

Postcondiciones:

- Hay una melodía disponible en el sistema, para poder ser visualizada y editada a través de la vista de segmentación.
- En el sistema hay una melodía disponible para ser reproducida.
- En el sistema hay una melodía disponible para ser almacenada en un archivo XML.
- Hay una melodía MIDI disponible en el sistema, para poder ser visualizada y editada a través de la vista tipo pianola.
- En el sistema hay una melodía MIDI disponible para ser reproducida.
- En el sistema hay una melodía MIDI disponible para ser almacenada en un archivo MIDI.
- En el sistema hay una melodía MIDI disponible para ser almacenada en un archivo XML.

Escenario principal de éxito:

1. El usuario selecciona la opción de menu: Analysis → Extract Melody
2. El sistema realiza el proceso de extracción de melodía, y su <<conversión a MIDI>> de forma simultánea.
3. Cuando el proceso termina, el sistema muestra un mensaje informativo al usuario.
4. El usuario pulsa el botón de aceptación, situado en el cuadro de mensaje mostrado por el sistema.

Includes:

- MIDI Convert (convertir a MIDI).

UC12

Caso de uso: MIDI Convert (convertir a MIDI).

Contexto: El usuario quiere obtener la melodía MIDI contenida en el audio original.

En este caso, el proceso de conversión a MIDI es realizado por el sistema de forma simultánea respecto del proceso de extracción de la melodía en su formato inicial. El formato inicial contiene valores de energía y f_0 para las notas que forman la melodía, en lugar de los valores MIDI cuantificados a partir de dichos valores iniciales.

Actores principales: Usuario, Sistema.

Precondiciones: El proceso de análisis ha sido realizado y el usuario ha seleccionado la opción de menú: Analysis Extract → Melody.

Postcondiciones: En el sistema hay una melodía MIDI disponible.

Escenario principal de éxito:

1. El sistema obtiene la representación MIDI de la melodía, de forma simultánea al proceso de extracción de la melodía en su formato inicial.

UC13

Caso de uso: View Segmentation (ver resultados del proceso de segmentación).

Contexto: El usuario quiere tener una vista que le informe de los resultados obtenidos en el proceso de segmentación.

Actores principales: Usuario.

Precondiciones: El proceso de extracción de melodía ha sido realizado.

Postcondiciones: –

Escenario principal de éxito:

1. El usuario selecciona la opción de menú: View → Segmentation.
2. El sistema muestra una vista de los datos de análisis vs. resultados de la segmentación para una determinada porción de la señal. Los límites temporales de las notas que componen la melodía se representan mediante líneas verticales.
3. El usuario utiliza la barra de scroll para avanzar y retroceder a lo largo de la señal.
4. El sistema actualiza la vista con los nuevos datos.
5. Se repiten 3 y 4 mientras el usuario lo desee.

UC14

Caso de uso: View Piano Roll (ver vista tipo pianola).

Contexto: El usuario quiere tener una vista tipo pianola, que represente la melodía MIDI resultante del proceso de conversión.

Actores principales: Usuario.

Precondiciones: El proceso de extracción de melodía ha sido realizado.

Postcondiciones: –

Escenario principal de éxito:

1. El usuario selecciona la opción de menú: View → Piano Roll.
2. El sistema muestra una vista de los datos de análisis vs. resultados de la conversión MIDI para una determinada porción de la señal. Los límites temporales de las notas que componen la melodía se representan mediante líneas verticales.
3. El usuario utiliza la barra de scroll para avanzar y retroceder a lo largo de la señal.
4. El sistema actualiza la vista con los nuevos datos.
5. Se repiten 3 y 4 mientras el usuario lo desee.

Extensiones:

* En cualquier momento, el usuario puede alternar las vistas relacionadas con los datos de análisis, pulsando los botones situados en el panel lateral izquierdo.

UC15

Caso de uso: Edit (edición de melodía).

Contexto: El usuario quiere modificar la melodía obtenida.

Actores principales: Usuario.

Precondiciones: La vista actual es la vista de segmentación, o la vista tipo pianola.

Postcondiciones:

– Los nuevos valores de melodía y melodía MIDI son mantenidos en memoria, en espera de que el usuario decida salvar o deshacer los cambios realizados.

Escenario principal de éxito:

1. El usuario pulsa con el botón izquierdo del ratón, sobre alguna de las líneas verticales que indican el inicio o fin de una nota.
2. El sistema guarda el estado actual de la melodía. El sistema cambia el aspecto del cursor del ratón, para indicar que el límite temporal representado por la línea vertical sobre la cual ha pulsado, puede ser modificado arrastrando el ratón a derecha o a izquierda. También se actualizan las etiquetas del panel inferior, para mostrar información relativa a la nota que está siendo editada.
3. El usuario arrastra el ratón a derecha o a izquierda.
4. El sistema actualiza la posición del límite temporal que está siendo modificado, y sincroniza las vistas de segmentación y pianola.
5. El usuario libera la pulsación ejercida sobre el ratón.
6. Se repiten 1 a 5 mientras el usuario lo desee.

UC16

Caso de uso: Undo (deshacer acción de edición).

Contexto: El usuario quiere deshacer la acción de edición anterior.

Actores principales: Usuario.

Precondiciones: La vista actual es la vista de segmentación, o la vista tipo pianola, y el tamaño de la pila de estados de edición es mayor que cero; es decir, existe alguna acción susceptible de ser deshecha.

Postcondiciones:

– Los valores de melodía y melodía MIDI que se mantienen en memoria en espera de que el usuario los valide o deshaga, se actualizan según los valores que se tenían en el estado anterior a la acción que ha sido deshecha.

Escenario principal de éxito:

1. El usuario pulsa con el botón derecho del ratón sobre el área de display.
2. El sistema muestra un menu emergente, entre cuyas opciones se encuentra la opción 'Undo'.
3. El usuario pulsa con el botón izquierdo del ratón sobre la opción 'Undo', en el menu emergente mostrado por el sistema en el paso 2.
4. El sistema desapila el estado anterior, por lo que el estado de la melodía pasa a ser el que tenía antes de haberse realizado la acción de edición que ha sido deshecha.

UC17

Caso de uso: Save (guardar cambios).

Contexto: El usuario quiere guardar los cambios aplicados a la melodía.

Actores principales: Usuario.

Precondiciones: La vista actual es la vista de segmentación, o la vista tipo pianola, y el tamaño de la pila de estados de edición es mayor que cero; es decir, existe algún cambio susceptible de ser guardado.

Postcondiciones:

– La melodía y la melodía MIDI se actualizan según los nuevos valores.

Escenario principal de éxito:

1. El usuario pulsa sobre el botón 'Save'.
2. El sistema actualiza la melodía y la melodía MIDI según los nuevos valores.
3. El sistema vacía la pila de estados.

Extensiones:

* De forma alternativa, el usuario puede llevar a cabo la acción 'Save' a través del menu emergente con el que se realiza la acción 'Undo' (ver UC16).

UC18

Caso de uso: Discard (descartar todos los cambios).

Contexto: El usuario quiere descartar todos los cambios realizados a partir del último 'Save'.

Actores principales: Usuario.

Precondiciones: La vista actual es la vista de segmentación, o la vista tipo pianola, y el tamaño de la pila de estados de edición es mayor que cero; es decir, existe algún cambio susceptible de ser descartado.

Postcondiciones:

– Los valores de melodía y melodía MIDI que se mantienen en memoria, pasan al estado en que se encontraban en el inicio, o después de haber realizado la última acción 'Save'.

Escenario principal de éxito:

1. El usuario pulsa con el botón derecho del ratón sobre el área de display.
2. El sistema muestra un menu emergente, entre cuyas opciones se encuentra la opción 'Discard'.
3. El usuario pulsa con el botón izquierdo del ratón sobre la opción 'Discard', en el menu emergente mostrado por el sistema en el paso 2.
4. El sistema restaura el estado de la melodía, de forma que pase al estado en que se encontraba después de haberse realizado la última acción 'Save'.
5. El sistema vacía la pila de estados.

UC19

Caso de uso: Play Fund Seg (reproducir frecuencia fundamental segmentada).

Contexto: El usuario quiere reproducir la frecuencia fundamental resultante del proceso de extracción de melodía, o de la posterior edición de la misma.

Actores principales: Usuario.

Precondiciones: El proceso de extracción de melodía ha sido realizado.

Postcondiciones: –

Escenario principal de éxito:

1. El usuario selecciona el botón de radio Fundamental SEG en el grupo Play.
2. El usuario pulsa el botón **Play**.
3. El sistema comienza la reproducción.

Extensiones:

- 3.2 Si la vista actual es la ofrecida por la opción de menu: View → Original audio, el sistema muestra al usuario una visualización dinámica de la sinuode correspondiente al tono puro generado por un oscilador, durante el proceso de reproducción.
- 3.2 Si el usuario pulsa el botón **Stop**, el proceso de reproducción finaliza.

UC20

Caso de uso: MIDI settings (ajuste MIDI).

Contexto: El usuario quiere establecer los parámetros para el entorno de reproducción MIDI.

Actores principales: Usuario.

Precondiciones: El grupo MIDI está activo (el sistema tiene soporte MIDI).

Postcondiciones:

- Hay un dispositivo MIDI seleccionado.
- Hay un programa (instrumento) MIDI seleccionado.

Escenario principal de éxito:

1. El usuario selecciona un dispositivo MIDI, un instrumento MIDI, o ambos mediante los controles combo box del grupo MIDI.

UC21

Caso de uso: Play MIDI (reproducir melodía MIDI).

Contexto: El usuario quiere reproducir la melodía MIDI resultado de la conversión, o de la posterior edición de la misma.

Actores principales: Usuario.

Precondiciones:

- El proceso de extracción de melodía ha sido realizado.
- Hay un dispositivo de salida MIDI activo en el sistema y está seleccionado.
- Hay un instrumento MIDI seleccionado.

Postcondiciones: –

Escenario principal de éxito:

1. El usuario selecciona el botón de radio MIDI en el grupo Play.
2. El usuario pulsa el botón **Play**.
3. El sistema comienza la reproducción.

Extensiones:

- 3.1 Si el usuario pulsa el botón **Stop**, el proceso de reproducción finaliza.

UC22

Caso de uso: Store MIDI (almacenar archivo MIDI).

Contexto: El usuario quiere almacenar un archivo MIDI en disco.

Actores principales: Usuario.

Precondiciones: Existe una melodía MIDI disponible.

Postcondiciones: El archivo MIDI queda almacenado en el disco.

Escenario principal de éxito:

1. El usuario selecciona la opción de menu: File → MIDI → Save MIDI file...
2. El sistema muestra un cuadro de diálogo para guardar el archivo.
3. El usuario selecciona la ubicación y escribe el nombre del archivo a guardar.
4. El usuario pulsa el botón de validación en el cuadro de diálogo.
5. El sistema almacena un archivo MIDI con el nombre indicado por el usuario.

UC23

Caso de uso: Save melody (almacenar melodía).

Contexto: El usuario quiere almacenar la melodía en un archivo XML.

Actores principales: Usuario.

Precondiciones: Existe una melodía disponible.

Postcondiciones: El archivo XML con la melodía queda almacenado en el disco.

Escenario principal de éxito:

1. El usuario selecciona la opción de menu: File → Melody → Save melody...
2. El sistema muestra un cuadro de diálogo para guardar el archivo.
3. El usuario selecciona la ubicación y escribe el nombre del archivo a guardar.
4. El usuario pulsa el botón de validación en el cuadro de diálogo.
5. El sistema almacena un archivo XML con el nombre indicado por el usuario.

UC24

Caso de uso: Save MIDI melody (almacenar melodía MIDI).

Contexto: El usuario quiere almacenar la melodía MIDI en un archivo XML.

Actores principales: Usuario.

Precondiciones: Existe una melodía MIDI disponible.

Postcondiciones: El archivo XML con la melodía MIDI queda almacenado en el disco.

Escenario principal de éxito:

1. El usuario selecciona la opción de menu: File → Melody → Save MIDI melody...
2. El sistema muestra un cuadro de diálogo para guardar el archivo.
3. El usuario selecciona la ubicación y escribe el nombre del archivo a guardar.
4. El usuario pulsa el botón de validación en el cuadro de diálogo.
5. El sistema almacena un archivo XML con el nombre indicado por el usuario.

A.3 Cuestionario de usabilidad

Voice2MIDI

Cuestionario de Usabilidad

Voice2MIDI es un sistema de conversión de voz a MIDI. La aplicación ofrece al usuario diferentes funcionalidades, como por ejemplo la visualización y edición de datos bajo distintos tipos de vista. Para realizar la conversión, el sistema emplea técnicas de análisis espectral, y su núcleo está construido haciendo uso de CLAM (C++ Library for Audio and Music : <http://www.iaa.upf.es/mtg/clam>).

Este cuestionario ha sido elaborado con la intención de obtener información, sobre la opinión de distintos tipos de usuario respecto de las capacidades y nivel de usabilidad del programa. Por este motivo, agradecer en primer lugar tu colaboración al ocupar parte de tu tiempo en rellenarlo.

Para completar el cuestionario, se te pedirá que realices una serie de acciones haciendo uso de la aplicación, para a continuación exponerte un conjunto de enunciados con varias opciones de respuesta, para que selecciones la que te parezca más conveniente marcándola con una cruz.

Por favor, una vez rellenado el cuestionario, envíalo por e-mail a:

ismael.mosquera01@campus.upf.es

A ser posible, envía el archivo comprimido. Ej. **v2m-cuestionario.zip**

En el subject (asunto) del mensaje escribe: cuestionario v2m

Datos personales

Nombre y apellidos: Edad: Ocupación:
--

Si tienes dudas sobre como realizar alguna de las acciones solicitadas, consulta la ayuda de la aplicación, la cual está disponible a través de la opción de menú: ? → **Help...**

Tarea 1 - cargar un archivo de audio

- Carga el archivo **audio_out1.wav**, el cual se encuentra en la carpeta **sound**.
- Reproduce el archivo cargado mediante el botón **Play**.

Tarea 2 – análisis

- Analiza la señal de audio.
- Una vez realizado el análisis, se tiene la opción de visualizar los datos obtenidos. Visualiza los datos resultantes del análisis en sus diferentes modos de visualización (**Full** y **Step by Step**). Consulta valores en cada uno de estos modos de visualización, pulsando con el botón izquierdo del ratón sobre el área de visualización, y comprueba si los dos tipos de vista están sincronizados.
- Selecciona la opción **Fundamental**, en el grupo de botones de radio **Play**, y reproduce la frecuencia fundamental de la señal obtenida en el análisis.

Tarea 3 – extracción de melodía

- Extrae la melodía contenida en la señal.
- Una vez extraída la melodía, se dispone de dos nuevos tipos de vista (**Segmentation** y **Piano Roll**). Estos tipos de visualización permiten editar los límites temporales de las notas de la melodía extraída, así como la posibilidad de eliminar notas de la misma. Por favor, consulta la

ayuda de la aplicación disponible a través de la opción de menú: ? → **Help...** para obtener información sobre la edición en estos modos de visualización.

- Selecciona la opción **Fundamental SEG** en el grupo de botones de radio **Play**, y reproduce la melodía obtenida tras el proceso de segmentación. Observa la diferencia entre la reproducción del tono fundamental, antes (opción Fundamental) y después de realizar el proceso de segmentación.
- Selecciona un dispositivo MIDI activo en tu sistema, en el cuadro combinado **MIDI Devices**.
- Selecciona un instrumento MIDI entre los disponibles en el cuadro combinado **MIDI Instruments**.
- Selecciona la opción **MIDI** en el grupo de botones de radio **Play**, y reproduce la melodía MIDI, la cual será interpretada usando el instrumento elegido.

Tarea 4 – edición

- Visualiza los datos en la vista tipo segmentación.
- Ajusta los límites temporales de las notas en caso de que sea necesario, comprobando que cada vez que se está modificando un determinado límite, el panel inferior muestra datos acerca de la nota a la cual corresponde dicho límite. En caso de que el límite sea compartido por dos notas, la información mostrada en el panel siempre se refiere a la nota que se encuentra a la derecha del límite.
- Alterna con la vista tipo piano roll, y comprueba si las vistas de segmentación y pianola están sincronizadas.
- Activa el menú emergente pulsando con el botón derecho del ratón sobre el área de visualización en cualquiera de las dos vistas (segmentación o pianola). Comprueba que se pueden deshacer acciones.
- Reproduce ahora nuevamente la melodía (Fundamental SEG), y la melodía midi (MIDI), y comprueba si la melodía reproducida incorpora los cambios realizados en la edición.

Tarea 5 – guardar resultado

- Guarda el resultado en un archivo MIDI.
- Comprueba que el archivo MIDI guardado es correcto, reproduciéndolo mediante alguna aplicación que soporte reproducción MIDI, como por ejemplo WinAmp o el Media Player de Windows.

Tarea 6 – registro mediante micrófono

Realiza esta tarea en caso de que dispongas de un micrófono conectado a tu computadora. Para el registro de la señal, canta ta, da, te, de, ti, di... tal como se hace en el caso de la anterior señal (audio_out1.wav). Esto es así por motivos técnicos respecto del proceso de segmentación, el cual se espera mejorar para el caso de la voz cantada normalmente, pero que en estos momentos debe realizarse bajo esta restricción.

- Registra una señal de audio, entonando una serie de notas.
- Visualiza la forma de onda de la señal registrada.
- Reproduce la señal registrada.
- Guarda la señal en un archivo wav.
- Reproduce el audio guardado usando otra aplicación (WinAmp, Media Player, etc.)
- Repite las tareas 2 a 5 para el audio registrado.

Por favor, señala con una cruz la opción que te parezca más adecuada en cada uno de los siguientes enunciados. Para ello, sustituye el recuadro por una x :

Ej. Normal. → x Normal.

En cada enunciado hay un espacio por si deseas añadir un comentario adicional.

1. – La realización de las tareas ha sido:

- Muy fácil.
- Fácil.
- Normal.
- Complicada.
- Muy complicada.

Comentario:

2. – El sistema de reproducción me parece:

- Muy bueno.
- Bueno.
- Aceptable.
- Malo.
- Muy malo.

Comentario:

3. – La calidad de los resultados que se obtienen en la conversión me parece:

- Muy buena.
- Buena.
- Aceptable.
- Mala.
- Muy mala.

Comentario:

4. – La información ofrecida por el sistema respecto a la función que realiza es:

- Muy completa.
- Completa.
- Aceptable.
- Escasa.
- Muy escasa.

Comentario:

5. – El sistema de visualización de datos me parece:

- Muy acertado.
- Acertado.
- Aceptable.
- Poco acertado.
- Nada acertado.

Comentario:

6. – El sistema de edición me parece:

- Muy acertado.
- Acertado.
- Aceptable.
- Poco acertado.
- Nada acertado.

Comentario:

7. – Representar las octavas mediante un código de colores me parece:

- Muy acertado.
- Acertado.
- Aceptable.
- Poco acertado.
- Nada acertado.

Comentario:

8. – El uso de la aplicación es:

- Muy intuitivo.
- Intuitivo.
- Aceptable.
- Poco intuitivo.
- Nada intuitivo.

Comentario:

9. – La documentación de ayuda de la aplicación me parece:

- Muy completa.
- Completa.
- Aceptable.
- Escasa.
- Muy escasa.

Comentario:

10. – El número y tipo de instrumentos MIDI disponibles me parece:

- Muy adecuado.
- Adecuado.
- Aceptable.
- Poco adecuado.
- Nada adecuado.

Escribe aquí los instrumentos que piensas que deberían añadirse:

Escribe aquí los instrumentos que piensas que deberían quitarse:

Comentario:

11. – La interfaz gráfica de usuario me parece:

- Muy adecuada.
- Adecuada.
- Aceptable.
- Poco adecuada.
- Nada adecuada.

Comentario:

12. – En general, la aplicación me parece:

- Muy buena.
- Buena.
- Aceptable.
- Mala.
- Muy mala.

Comentario:

En caso de haber tenido problemas en la realización de las tareas, o debidos a fallos en la aplicación, escríbelos aquí:

Escribe aquí los aspectos que no hayas entendido, o te parezca que se encuentren fuera de lugar en el contexto de la aplicación:

Por favor, escribe aquí tu impresión y sugerencias acerca de la aplicación:

A.4 Instrucciones para la compilación del programa

A continuación, se indican los pasos a seguir para la construcción de la aplicación a partir del código fuente, para las plataformas windows y linux, utilizando los compiladores Visual C++ de Microsoft, y gcc de GNU respectivamente.

Como ya se ha comentado en un capítulo anterior, CLAM ofrece un entorno amigable mediante el cual organizar los proyectos. El código se encuentra organizado de forma que para realizar la compilación se necesite una intervención mínima por parte del usuario. Seguidamente, se enumera la secuencia de acciones que debe completarse para construir el ejecutable en ambas plataformas.

Linux

1. Descomprimir el archivo `Voice2MIDI.tar.gz`

```
gunzip Voice2MIDI.tar.gz
tar -xf Voice2MIDI.tar
```
2. Copiar la carpeta `Voice2MIDI` en el primer nivel de directorios de CLAM.
 Ej. `CLAM/Voice2MIDI`
3. Situarse en el directorio `build` dentro de `Voice2MIDI`
 Ej. `cd CLAM/Voice2MIDI/build`
4. Ejecutar el comando:

```
make CONFIG=release
```
5. Esperar a que termine el proceso de compilación y enlace.
6. Ejecutar la aplicación.

```
./Voice2MIDI
```

Windows

1. Descomprimir el archivo `Voice2MIDI.zip`
2. Copiar la carpeta `Voice2MIDI` en el primer nivel de directorios de CLAM.
 Ej. `CLAM\Voice2MIDI`
3. Situarse en el directorio `build` dentro de `Voice2MIDI`
 Ej. `cd CLAM\Voice2MIDI\build`
4. Construir el archivo de proyecto ejecutando el comando:

```
..\..\build\srcdeps\srcdeps settings.cfg
```
5. Abrir el archivo de proyecto con el VC.
6. Construir la aplicación.

****Nota:** Si se desea compilar una versión release, desactivar la expansión de funciones inline en la parte de optimizaciones en las propiedades del proyecto. El tener activa la expansión de funciones inline causa anomalías en las acciones de edición.

Bibliografía y referencias web

SUNDBERG, Johan. The Science of the Singing Voice.
Northern: Illinois University press, 1987

BERMUDEZ, Jesús. Reconocimiento de Voz y Fonética Acústica. Madrid: Rama, 2000

ROADS, Curtis. The Computer Music Tutorial.
Cambridge, Massachusetts: The MIT Press, 1996

ZÖLZER, Udo. DAFX – Digital Audio Effects. Chichester: John Wiley, 2002

JORDÀ, Sergi. Guía monográfica del audio digital y el MIDI. Madrid: Anaya Multimedia, 1997

STROUSTRUP, Bjarne. El Lenguaje de Programación C++. Madrid: Addison Wesley, 2002

AHO, Alfred V. Compiladores Principios, técnicas y herramientas.
México: Addison Wesley, 1990

DALHEIMER, Matthias Kalle. Programming with Qt. Beifimng: O'Reilly, 2002

WRIGHT, Richard. OpenGL SuperBible. Waite Group Press, 1999

LARMAN, Craig. Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design. Upper Saddle River, N.J. Prentice Hall PTR, 2001

CLAM

<http://www.iaa.upf.es/mtg/clam>

Qt

<http://www.trolltech.com>

FLTK (Fast Light Tool Kit)

<http://www.fltk.org>

Archivos MIDI

<http://www.borg.com/~jglatt/tech/midifile.htm>

<http://www.sfu.ca/sca/Manuals/247/midi/fileformat.html>

Portmusic (portmidi)

<http://www-2.cs.cmu.edu/afs/cs.cmu.edu/project/music/web/portmusic>

ALSA (Advanced Linux Sound Architecture)

<http://www.alsa-project.org>

pthread (librería de hilos de ejecución)

<http://www.llnl.gov/computing/tutorials/workshops/workshop/pthreads/MAIN.html>

C++ (artículo by Bjarne Stroustrup)

<http://www.research.att.com/~bs/whatis.pdf>

Relacionados con procesamiento de señal y segmentación

<http://www.norsig.no/norsig2002/Proceedings/papers/cr1047.pdf>

http://web.media.mit.edu/~moo/thesis/YEK_thesis.pdf

<http://citeseer.ist.psu.edu/cache/papers/cs/3390/http:zSzzSzwww.nzdl.orgzSzpublicationszSz1996zSzRJM-LAS-IHW-Sig-Proc.pdf/mcnab96signal.pdf>

Sistemas QBH (Query-by-Humming). Búsqueda de canciones por tarareo.

<http://www.dlib.org/dlib/may97/meldex/05witten.html>

<http://web.media.mit.edu/~chaiwei/qbhweb>

<http://www.sonoda.net/echo/index.html>

<http://www.ipem.ugent.be/MAMI/Public/Data/QbVExperiment>