27
25

UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

FACULTAD DE CIENCIAS

MULTIMEDIA: DESARROLLO DE APLICACIONES BAJO AMBIENTE WINDOWS

T E S I S

QUE PARA OBTENER EL TITULO DE

M A T E M A T I C A

P R E S E N T A :

MARIA DEL CARMEN RAMOS NAVA

DIRECTORA DE TESIS: MAT. ANA LUISA SOLIS GONZALEZ COSIO.

1996

VNIVERSIDAD NACIONAL
AVPNOMA DE
MEXICO

M. en C. Virginia Abrín Batule
Jefe de la División de Estudios Profesionales de la
Facultad de Ciencias
P r e s e n t e

Comunicamos a usted que hemos revisado el trabajo de Tesis:

Multimedia: Desarrollo de aplicaciones bajo ambiente windows.

realizado por María del Carmen Ramos Nava

con número de cuenta 8408207-5 , pasante de la carrera de Matemáticas

Dicho trabajo cuenta con nuestro voto aprobatorio.

Atentamente

Director de Tesis Propietario	Mat. Ana Luisa Solis González Cosio	
Propietario	M. en C. Ma. Guadalupe Elena Ibarguengoitia González	
Propietario	M. en C. Gustavo Márquez Flores	
Suplente	M. en C. Héctor Perales Valdivia	
Suplente	Fís. Jesús Gutiérrez García	

Consejo Departamental de Matemáticas
M. en C. ALEJANDRO BRAVO MOJICA

Agradecimientos

A mi madre Epifania
y mis hermanos Omar
Víctor, Gerardo y Eduardo
por brindarme siempre
su cariño y apoyo.

A mi asesora Ana Luisa Solis
por el gran apoyo
y conocimientos que de ella recibí.

Contenido

Introducción 1

Capítulo 1 Multimedia 3
1.1 Definición 3
1.2 Creación de aplicaciones 5
1.3 Areas de aplicación 6

Capítulo 2 Requerimientos para multimedia 9
2.1 Demanda de datos 10
2.2 Hardware multimedia 11
2.2.1 CPU 11
2.2.2 El sistema RAM 12
2.2.3 Despliegue de video 12
2.2.4 Sistema de audio 14
2.2.5 Almacenamiento 15
2.2.6 CD-ROM 15
2.2.7 Coprocesadores 16
2.2.8 Tarjetas especiales 17
2.3 Software 20
2.3.1 Sistema operativo 21
2.3.1.1 Microsoft Windows 23
2.3.1.2 Servicios multimedia 24
2.3.2 Sistemas de autoría 28
2.3.3 Programas de aplicación 32

Capítulo 3 Media Control Interface (MCI) 34
3.1 La arquitectura MCI 35
3.2 Las interfaces de programación MCI 36
3.2.1 La interfaz comando-cadena 36
3.2.2 La interfaz comando-mensaje 36
3.3 Conjunto de comandos MCI 37
3.4 Clasificación de los dispositivos MCI 38
3.5 Tipos de dispositivos estándar MCI 39
3.6 Nombres de dispositivos 40
3.7 Utilizando la interfaz comadno-cadena 41
3.8 Utilizando la interfaz comando-mensaje 42
3.8.1 Mensajes de comandos 43

Capítulo 4 Programación de medios en ambiente windows 45
4.1 Objetos Multimedia 45
4.2 Objetos activos 46
4.2.1 Sonido 46
4.2.1.1 WAVE 46
4.2.1.2 CD-ROM 61
4.2.2 Video 66
4.3 Objetos pasivos 70
4.3.1 Imágenes 70
4.3.1.1 Estructuras bitmap de Windows 72
4.3.1.2 Despliegue de bitmaps 75

Capítulo 5 Biblioteca de clases MCI 79
5.1 Diseño 79
5.2 Dispositivos simples MCI 83
5.3 Dispositivos compuestos 84
5.4 Compilación 85
5.5 Ejemplo de aplicación 86

Conclusiones 93

Apéndice A Sumario de comandos 95

Apéndice B Código de la biblioteca de objetos MCI 98

Bibliografía 117

Introducción

La Multimedia ha tomado un gran auge debido a su característica de poder integrar en aplicaciones interactivas, diferentes medios, como son video, sonido, animaciones y gráficos. Esto ha dado como resultado ambientes dinámicos y agradables para el usuario, que abarcan un gran rango de áreas de aplicación, que van desde el entretenimiento y la educación, hasta el adiestramiento y la distribución de información.

El desarrollo de multimedia puede hacerse en diferentes plataformas como son estaciones de trabajo o computadoras personales, siempre y cuando se cuente con el software y hardware apropiado para ello.

El presente trabajo está enfocado a máquinas personales PC y ambiente windows, dado que es una de las plataformas más utilizadas en el desarrollo de este tipo de aplicaciones.

En esta plataforma existen diferentes formas de desarrollar aplicaciones multimedia, como el que utiliza software de autoría que permite crear las aplicaciones de una forma más fácil pero limitada, debido a que el programador se tiene que conformar con las funciones

y herramientas que le proporcione el sistema con el que se esté trabajando y en la mayoría de los casos estos sistemas no permiten integrar código propio.

Así la propuesta es que el desarrollo de multimedia sea a partir de programación en lenguaje C++ con Windows, accediendo a los medios de audio, video y animación, a través de la interfaz MCI *(Media Control Interface)*. Esta forma de programación nos permite crear aplicaciones independientes de dispositivos con la mayor flexibilidad en el desarrollo. La independencia de dispositivos es una característica que proporciona MCI, y la flexibilidad la tenemos ya que a este nivel de programación, cualquier algoritmo que se requiera puede ser implementado, además de que el acceso al API de Windows es directo.

Los objetivos de este trabajo son mostrar los elementos necesarios de software y hardware, para el desarrollo de una aplicación multimedia; así como las opciones de programación de los diferentes medios, a través de las extensiones Multimedia Windows; entre las que se incluye la programación a través de MCI y el desarrollo de una herramienta de software que facilite la programación de los medios a través de la misma interfaz. Esta herramienta consiste de una biblioteca de objetos MCI, que encapsule características de los medios, permitiendo programarlos en una forma sencilla, pero sin perder eficiencia.

El trabajo se dividió en 5 partes: la primera da una descripción de lo que es multimedia y sus principales aplicaciones. La segunda parte describe cuales son los elementos de software y hardware, dentro de ella se explican algunos sistemas de autoría actualmente utilizados; en la tercera parte se describe el diseño de MCI, para entender la forma en que proporciona la independencia de dispositivos; en la cuarta parte se muestran las formas en que se pueden realizar la programación de los medios a partir de las extensiones multimedia desde un programa en C++; y por último, en la quinta parte, se presenta el diseño e implementación de la biblioteca de objetos MCI, mostrando su utilización en el desarrollo de un programa de video interactivo que hace una navegación por la biblioteca de la Facultad de Ciencias.

Capítulo 1

Multimedia

1.1 Definición

Se considerá un sistema multimedia como aquél que es capaz de proporcionar la presentación de texto, gráficos, animación, reproducción de audio real, escenas en pantalla completa de video real e imágenes fijas de calidad fotográfica, todos ellos en formato digital, brindando la habilidad de presentarlos en una forma integrada e interactiva.

Multimedia digital

El tener todos los medios de texto, audio, gráficos, animación y video en formato digital ofrece la ventaja de que puedan ser tratados como cualquier otro archivo digital. El video y el audio pueden ser capturados por la computadora y guardarse en disco duro como archivos. Esta información puede ser editada: las imágenes pueden ser recortadas de tal forma que partes innecesarias sean eliminadas; los archivos de video digital puedan hacerse más brillantes u obscuros; las pistas de sonido puedan ser integrados al video; se pueden escribir con el software efectos especiales -acercamientos, desvanecimentos, etc;- que pueden aparecer sin postproducción costosa; gráficos o textos pueden ser incluidos en video o imágenes fijas. De igual forma se puede manipular cualquier parte de la presentación multimedia, aprovechando la capacidad de procesamiento digital de la computadora, con la ventaja de que como todo el trabajo lo realiza la máquina, no se requieren componentes externos. Además se puede almacenar y distribuir las aplicaciones sobre medios estándar tales como discos flexibles, discos duros, cintas digitales, CD-ROM y redes de computadoras.

Dado que los sistemas de audio, receptores de televisión, cámaras de video, reproductores de video laser, son análógicos, no podrían ser integrados al sistema de tal manera que trabajarían como hardware separado y no se benefician del almacenamiento digital.

Interactividad

Otro de los elementos importantes de multimedia es la interactividad, lo cual significa que el usuario tiene el control de lo que verá y escuchará como resultado de las opciones y decisiones que realize. En las aplicaciones multimedia se deben crear ambientes interactivos en los cuales el usuario tenga el control y se encuentre cómodo con este ambiente.

1.2 Creación de aplicaciones

Una aplicación multimedia puede ser tan simple como aumentar una base de datos con imágenes, incorporar una función de ayuda en audio en uno o más lenguajes, hasta un simulador completo de instrucción de una misión de vuelo muy peligrosa y costosa.

El desarrollo es similar al de una película o libro. El creador decide sobre el contenido: un problema de adiestramiento, una presentación de negocios con construcción interactiva o un juego de aventuras con personajes y escenas de una serie de televisión. El contenido deberá ser creado y producido. Entonces el software es escrito para manejar el contenido. El software y el contenido conforman una aplicación multimedia.

Al proceso de desarrollo que consiste en integrar los diversos medios, se llama *autoría* y puede ser muy sencillo, o complicado, dependiendo de lo que se espera como resultado. El tipo de aplicación que se quiera crear, depende de los objetivos, el mercado y los recursos. Si la aplicación es muy sofisticada entonces es elaborada por un extenso equipo de expertos, quienes realizan las siguientes actividades:

- Producción y postproducción de audio
- Producción y postproducción de video
- Programación de computadoras
- Diseño artístico
- Arte gráfico

1.3 Areas de aplicación

Capacitación

La capacitación por medio de video interactivo ha sida utilizada con mucho éxito. Varios estudios han demostrado que esta forma de capacitación incrementa la retención, decrementa el costo y minimiza el tiempo requerido para las sesiones de adiestramiento.

Los programas de entrenamiento por computadora, involucran simulaciones que guían a los usuarios a través de secuencias y procedimientos de pasos interactivos. Estos sistemas deben proporcionar situaciones realistas y un alto grado de interacción.

Las simulaciones basadas en gráficas son inadecuadas para sesiones de vida real y no son capaces de incorporar secuencias de filmes reales o video mostrando el tema de adiestramiento, y dado que el realismo es necesario en campos como la medicina y la aviación, un multimedio digital tiene un gran potencial en simulaciones para entrenmmiento, debido a su habilidad de incorporar medios como el video.

Por ejemplo, las capacidades de división de pantalla permite a los usuarios selectivamente comparar imágenes fijas o en movimiento, para estudiar rangos de temas desde cocinar, hasta procedimientos de laboratorio. Las gráficas de alta calidad pueden ser combinados con video para permitir a un estudiante de reparación de automóviles ver y escuchar la explicación de como quitar la bomba de gasolina, así como examinar y manipular un modelo de una bomba real en otra sección de la pantalla. El estudiante podrá seleccionar en la pantalla de un catálogo de partes y obtener información textual acerca de modelos, costos y diponibilidad de bombas de gasolina.

Las simulaciones son utilizadas en ambientes donde es imposible filmar todo. Por ejemplo, animaciones de alta velocidad pueden ser usados para crear un mundo interactivo microscópico o el modelo de la anatomía humana. Técnicas de laboratorio muy caras

o peligrosas pueden ser simuladas antes de que sean realizadas, minimizando costos y resultados potencialmente dañinos en un ambiente médico o industrial.

En el entrenamiento militar se hace un uso extenso de la simulación como es el caso del entrenamiento de pilotos, el adiestramiento en mantenimiento de submarinos.

Por ejemplo el Southwest Research Institute de San Antonio, Texas desarrolló una aplicación que trabaja con un robot cuya tarea es el pintado de láminas. Esta aplicación es del tamaño de un hangar de aeroplanos y es utilizado para pintar las láminas de los aeroplanos. El sistema de entrenamiento que va con el robot utiliza video de gran tamaño, gráficos, audio y texto, en una aplicación que enseña al operador como usar el robot y como enseñar al robot nuevos movimientos.

Obtención y distribución de información

Los kioskos interactivos multimedia localizados en sitios de información alrededor de algún sitio pueden ser utilizados por los visitantes para aprender acerca de las atracciones locales, flora, fauna y facilidades recreativas cuando se viaja. Este tipo de kioskos son actualmente utilizados en Florida y en aeropuertos de Francia.

Estos sistemas fueron la inspiración para una nueva forma de consumo de productos. Los kioskos pueden ser colocados en cualquier lugar, mueblerías, librerías, tiendas de abarrotes, etc. Y el usuario puede obtener información de los productos a través de imágenes, y videos, y en algunos casos, ordenar su compra desde ahí.

Otra forma de manejo de información es la utilizada en la aplicación "Design & Decorate", co-desarrollado con Videodisk Publishing, Inc. New York. Esta aplicación permite a los usuarios diseñar una sala a la medida, seleccionar el tamaño y la forma del cuarto, los muebles, la pintura y el papel tapiz, combinar éstos con accesorios de habitación opcionales. Esta nueva sala, recién creada, puede ser vista desde una variedad de ángulos determinados por los comandos. Además pueden ser proporcionados catálogos en línea acerca de muebles y fábricas.

Este tipo de aplicaciones son utilizados como una herramienta de venta en muchas otras áreas como ventas de automóviles donde las alternativas de color pueden ser mostradas.

Educación

Una de las primeras aplicaciones de multimedia en este campo fue "Palenque". Un recorrido a través de las ruinas mayas controlado por una palanca de juego. Los niños y adultos que usaban esta aplicación fueron capaces de caminar alrededor de Palenque y podían acceder la información desde personajes como C.T. Granville, un joven, que era acompañado, a lo largo del recorrido, por un arqueólogo y un botánico; teniendo a la mano la información acerca de plantas, animales, historia maya, y mapas del sitio.

Otro grupo en Canada, en la Biblioteca Lindsay, en conjunción con Pixel Productions, están utilizando multimedia para documentar la historia de la ciudad del mismo nombre. Los miembros de la biblioteca están coleccionando fotografías históricas, video, audio y produciendo nuevos medios para incluir en la base de datos de la ciudad. Está aplicación está dibujada sobre las experiencias y las memorias de los personajes de la ciudad, creando una pieza patrón que preservará la historia de Lindsay para todos sus habitantes.

Entretenimiento

Juegos como golf, vuelos, aventura y misterio pueden ser desplegados con acción de video real, mientras mantienen la versatilidad de la animación. Por ejemplo un programa de simulación de vuelo, co-desarrollado con Activision of Mountain View CA, combina imágenes reales y respuestas interactivas a través del uso de video. El piloto vuela un World War II Spitfire sobre una parte del territorio inglés, complementada con sonidos de un Spitfire real y video de construcciones texturizadas.

Capítulo 2

Requerimientos para multimedia

Hay numerosas familias de computadoras en el mercado, incluyendo las máquinas IBM-compatibles, la familia Apple Macintosh, estaciones SUN y Silicon, las computadoras NEXT y otras. Todas son adecuadas para multimedia cuando se equipan con la suficiente capacidad de velocidad y almacenamiento. Sin embargo, en esta tesis solo se tratará con sistemas IBM-compatibles PCs, por ser el equipo con el que se contaba al momento de comenzar el trabajo, en el Laboratorio de Graficación y Multimedia de la Facultad de Ciencias.

Antes de describir los elementos necesarios para crear sistemas multimedia, es importante establecer cuales son las demandas que los justifican.

2.1 Demanda de datos

La mayor demanda de recursos se crea de la necesidad de presentar video dinámico, debido a que una computadora multimedia debe realizar despliegues rápidos de imágenes complejas (alta resolución y gran número de colores); es decir, el sistema debe ser capaz de traer rápidamente la información del despliegue desde donde se almacena, procesarlo y pasar esto al subsistema de despliegue de video para su presentación. Algunas veces esto requiere hacerse tan rápido como 30 veces por segundo, para crear una representación adecuada del movimiento. Además, los archivos de datos de video pueden ser tan grandes que llegan a ser imprácticos sin el uso de compresión de video, el cual utiliza el procesamiento de la computadora para remover la redundancia de las imágenes de video, de tal modo que reduce los datos requeridos. Esto incrementa substancialmente la carga de procesamiento, lo cual lleva a usar un hardware especial para despliegue de video.

La habilidad para desplegar detalles finos en la imágen depende de la resolución de la máquina y la cantidad de colores que pueden ser desplegados por cada pixel. Por ejemplo, una pantalla de despliegue VGA tiene 640 x 480 pixeles, dando una resolución de 307,200 pixeles por imágen y si se manejan 2 bytes (16 bits) para color por cada pixel, entonces la imágen VGA de 16 bits, ocuparía 614,400 bytes (640x480x2), manejando 65 536 colores.

Para producir un video fluido usando imágenes de computadora, se tendría que desplegar imágenes nuevas al menos 15 veces por segundo (30 veces por segundo es mejor). Para la imágen de 16 bits VGA descrita arriba, 15 imágenes por segundo trabaja a aproximadamente a 8.78 megabytes de datos por segundo, alrededor de 527 megabytes por minuto. Estos números actualmente sobrepasan la capacidad de una computadora personal, por lo que no es posible desplegar video solamente mostrando una serie de imágenes traídas de disco.

Lo anterior es solo para la imágen, sin incluir el audio. Si nos basamos en los datos de un sistema de audio digital Compact Disc (CD) usado para sistemas de audio caseros, que tienen una tasa de 150.000 bytes por segundo, ó 8.58 megabytes por minuto, nos damos

cuenta que es aproximadamente igual a lo indicado antes para el manejo de vid eo. Utilizar esta clase de audio, con video digital, podría doblar el número de datos.

Así, las computadoras personales actuales no están equipadas para transferir o guardar esta cantidad de datos, los del video y audio digital tienen que ser comprimidos en tal forma que sean lo suficientemente pequeños para ser guardados y procesados por la computadora.

2.2 Hardware multimedia

2.2.1 CPU

Un sistema multimedia debe beneficiarse de la rápidez del CPU. Por lo menos se debe considerar un microprocesador 386 ó 486.

En la arquitectura básica de la computadora sólo una unidad en el bus, que conecta a todos los componentes puede operar en un tiempo. Lo mismo se aplica al CPU, este sólo hace una cosa a la vez. Sin embargo, el CPU y el bus operan a millones de ciclos por segundo, lo cual hace parecer que la mayoría de las actividades ocurren simultáneamente, cuando en realidad ocurren en secuencia. Por ejemplo, la PC puede verse como que esta generando sonido y actualizando el despliegue al mismo tiempo; de hecho, esto lo hace secuencialmente, de aqui la importancia de la velocidad del procesador.

2.2.2 El sistema RAM

La RAM (*random access memory*) es la parte que proporciona almacenamiento para datos temporales y programas usados por el CPU; entre mayor RAM es mejor. La mayoria de las 386 y sistemas mas grandes soportan al menos 16 megabytes de RAM, y muchos pueden tener más. Para un uso multimedia, se requieren al menos 8 Mb de RAM en un sistema que será para desarrollo, debido a los requerimientos del software para desarrollo, y no menos de 6 Mb para un sistema que sólo será usado para presentaciones.

2.2.3 Despliegue de video

El despliegue de video en todas las PC utiliza mapeo de memoria, en la cual, una región de memoria guarda una lista de todos los pixeles para ser desplegados en la pantalla. Para crear el despliegue, el adaptador de video lee, a través de esta memoria, a una velocidad muy rápida y convierte los valores de los pixeles a señales que maneja el monitor. Este proceso requiere una gran actividad de memoria para mover los datos para el refrescamiento de la pantalla, regularmente de 60 a 70 veces de la pantalla completa por segundo para prevenir el temblor de la pantalla.

Por ejemplo, un despliegue de 640x480 de 16 colores VGA, que actualiza 60 veces por segundo, accesa 9.22 millones de bytes por segundo. El bus del sistema no podría manejar esta cantidad aunque sólo hiciera esto. Sin embargo, los adaptadores de despliegue siempre incluyen su propia memoria para guardar la información de la pantalla, y tienen circuitos separados para refrescar el monitor desde esa memoria sin hacer uso del bus del sistema. Por supuesto, el bus es aún usado cuando el CPU tiene que actualizar la información que será desplegada.

Otra especificación importante del depliegue de video es el número de colores que pueden ser deplegados. Este esta directamente relacionados al numero de bits de datos

asignados a cada pixel.

La mayoría de los despliegues de video de 8 bits por pixel usan los "lookup table" o paletas de color para proporcionar flexibilidad al seleccionar los 256 colores específicos que serán disponibles para los pixeles. La paleta es una tabla que se guarda en el adaptador del video, el cual contiene 18 bits o hasta 24 bits de descripción de los 256 colores diferentes que pueden ser representados por los 256 valores de un pixel de 8 bits. Cuando cada pixel es desplegado, su valor es usado como un índice dentro de la paleta, leyendo la información de 18 o 24 bits de color, que se desplegará. La ventaja de esto es que se puede definir a la paleta para tener los mejores 256 colores para reproducir la imágen que se tiene. En principio, se puede cambiar la paleta para cada imágen.

El método de la paleta además de manejar sólo 256 colores tiene otra desventaja. Se debe hacer el procesamiento de las imágenes cuando se capturan para crear su paleta de color y guardarla junto con la imágen. Esta es la mejor aproximación, porque una imágen natural tiene cientos de colores, los cuales tienen que ser ajustados a 256, que son los disponibles. Pero la dificultad real es que únicamente se puede tener una paleta para la pantalla completa, así solo puede contener los colores para una imágen . Por lo tanto, presenta muchas dificultades el construir una pantalla conteniendo mas de una imágen. Si se quieren usar dos imágenes, se tienen que procesar estas al mismo tiempo y construir una paleta común que tenga la mejor aproximación de los colores de ambas imágenes. Por lo tanto el manejo de la paleta de color no es suficiente para aplicaciones multimedia.

El despliegue de imágenes reales requieren al menos 16 bits por pixel. Los sistemas de 16 bits por pixel tienen 32, 768 o 65,536 colores dependiendo del uso del bit mas alto. Ambos pueden reproducir escenas naturales, aunque ocasionalmente se pueden ver algunos problemas sobre escenas que tienen colores muy parecidos, como el acercamiento de la cara de un bebe. Para evitar este efecto por completo se tiene que utilizar 24 bits por pixel.

2.2.4 Sistema de audio

El audio o la capacidad de sonido de una computadora, regularmente consiste de una pequeño manejador interno llamado *speaker* cuya salida es de 1 bit. Una cantidad asombrosa de sonidos puede ser producida por la rápidez de hacer un switch a esta salida. Sin embargo, los sonidos son artificiales , y no se acercan a algo real, es decir, a un sonido natural. La situación es la misma que para el video. Se necesitan más bits para reproducir sonido natural.

Se logra grabar digitalmente audio natural por el muestreo del audio analógico. Muestrear es una técnica que repetidamente convierte el valor de una onda análogica a una número digital. Esto es llamado ADC (*analog to digital conversion*) o digitalización. Para una buena calidad de audio la tasa de muestreo, es decir que tan rápido se toman las muestras, debe ser de 20.000 muestras por segundo o más. La exactitud de cada muestra, expresada en número de bits por muestra, debera ser de 14 a 16 bits para un audio de alta fidelidad. Cuando el audio digital es ejecutado, se realiza el proceso inverso (*digital to analog conversion DAC*). Un sistema multimedia necesita de esta capacidad de audio digital.

Otra capacidad de audio, el cual se aplica a la producción de música, es llamada MIDI (Musical Instrument Digital Interface). MIDI es una comunicación estándar digital desarrollada por la industria de instrumentos musicales para transmitir comandos entre teclados u otros controladores de música y unidades de producción de sonidos tales como los sintetizadores. Los comandos MIDI , dado que son digitales, pueden ser guardados por una computadora y posteriormente son ejecutados en uno o más instrumentos para recrear la música. Algunos sistemas multimedia usan sintetizadores para crear música en esta forma. Dependiendo de la habilidad y capacidad de los sintetizadores usados, MIDI es capaz de producir toda una música orquestal.

2.2.5 Almacenamiento

Los medios de almacenamiento pasan todos sus datos a través del bus del sistema para cargarlos dentro del RAM, o la memoria de despliegue de video. La velocidad de los dispositivos de almacenamiento es determinado por la velocidad a la que ellos pueden leer desde su medio de almacenamiento. Esta operación requiere de un gran procesamiento del CPU. De esta forma, el número de datos que realmente pueden ser accesados disminuye. Por ejemplo, las tasas de datos máximas de disco duro son alrededor de 500,000 bytes/segundo cuando un bloque contiguo de datos se lee sin búsquedas involucradas. Si se incluyen la acción del sistema operativo y el bus del sistema, difícilmente se mantiene un tasa de datos de disco duro promedio o más de 150,000 bytes/segundo .

Se necesita una gran capacidad de almacenamiento, ya que multimedia requiere mucho espacio. El video puede requerir arriba de 10Mbytes por minuto, e imágenes individuales fotográficas están en el rango de 50 - 100 Kb, incrementándose, y haciendo que un disco duro de 100 Mbytes sea pequeño. Por lo tanto se debe escoger cuidadosamente la capacidad del disco duro, considerando las necesidades de almacenamiento de multimedia antes de decidir.

2.2.6 CD-ROM

El disco compacto o CD_ROM (*Compact Disc Read Only Memory*) es un importante medio de almacenamiento para multimedia. Basado en el hardware originalmente desarrollado para reproductores de audio digital, el CD-ROM proporciona arriba de 680 Mb de almacenamiento en un disco compacto de una cara de 12 cm; dado que proviene del audio digital, el hardware del CD-ROM no es caro. Sin embargo, éste es solo un medio de lectura, lo cual significa que una vez que el CD-ROM ha sido producido, los datos sobre él no pueden ser cambiados. Esto es una ventaja y una desventaja: Los datos son permanentes y no pueden ser modificados fácilmente, pero no se pueden añadir nuevos datos a

él. Dado que siempre se requieren almacenar nuevos datos, una sola unidad de CD-ROM no sirve para todo el almacenamiento de datos que se requiere. A pesar de la desventaja, el CD-ROM es hasta hoy el mas importante medio de bajo costo para la distribución de grandes cantidades de datos a bajo costo.

El CD-ROM logra su gran capacidad de almacenamiento al usar una tecnología óptica.

Todas las unidades ópticas tienen como característica un tiempo de acceso grande(el tiempo para encontrar una pieza particular de datos y empezar a leer a partir de ahí) a diferencia de las unidades magnéticas. Esto por la necesidad de que una unidad óptica, para leer datos, debe estar seguro en que pista está, mientras que una unidad magnética conoce la localización de pista por un simple posicionamiento mecánico. El resultado es que los unidades magnéticas hoy en día accesan sus datos, 10 o más veces más rápido que las unidades ópticas. El tiempo de acceso es importante para las aplicaciones multimedia porque éste contribuye a el retardo que puede ocurrir entre la opción que el usuario selecciona y que la acción comience a parecer en la pantalla.

Pero a pesar de su gran tiempo de acceso, la capacidad de almacenamiento del CD-ROM lo hace un elemento importante para multimedia.

2.2.7 Coprocesadores

Ciertas tareas en un sistema PC ocurren repetidamente y pueden ser aceleradas al proporcionar hardware dedicado, en lugar de ejecutarlas todas por software. El hardware extra es usualmente en la forma de chips de circuitos integrados diseñados para tal propósito: ellos son referidos como coprocesadores .

El coprocesador mas común es uno usado en las máquinas que corren programas que realizan grandes cantidades de operaciones aritméticas: un coprocesador matemático. Los coprocesadores matemáticos son usados para aplicaciones de diseño asistido por computadoras, hojas de cálculo, y modelado de gráficos, entre otros. Para multimedia, las necesidades de video, audio y animación pueden ser mejoradas con ayuda de coprocesadores

especiales.

2.2.8 Tarjetas especiales

Hasta aquí se ha discutido los componentes estándar que se pueden utilizar en multimedia. Sin embargo, los adaptadores de despliegue en las computadoras estándar y los sistemas de sonido no soportan audio de alta calidad y capacidades de video real. Esta es la razón por la que la mayoría de las PC's proporcionan tarjetas adicionales, existiendo muchas y diferentes tarjetas en el mercado. Desafortunadamente, no hay estandarización del hardware de audio y video, y cada producto requiere sus propios manejadores de software y algunas veces software de alto nivel también. Es aqui donde se ve la gran utilidad del software MCI que se discutirá posteriormente, ya que proporciona una interface genérica para una variedad de tarjetas.

A pesar de que hay un gran número de sistemas de software en el mercado de compresión de video (AVI de Microsoft, Photomotion and Ultimotion de IBM, y Apple QuickTime). Aún, con la disponibilidad de los CPUs más rápidos, pierden resolución, número de colores, velocidad de datos ó de cuadros. Así, a causa de su simplicidad, el software de compresión de video no compite realmente con sistemas de coprocesadores de video los cuales pueden proporcionar además de pantallas completas, color real y video de 30 cuadros por segundo.

Si se esta ensamblando un sistema para desarollar aplicaciones multimedia y no solo para reproducirlas, se necesita otro equipo para la entrada de video y audio dentro de la computadora. El audio y video generado con equipo de video y audio es analógico, como el utilizado para la televisión. Dependiendo de los niveles de calidad en los cuáles se necesita trabajar, se puede emplear desde un equipo de audio y video casero tales como VCR, o reproductores de cassetes; hasta un equipo de audio y video de uso profesional, el cual, por supuesto, es más caro.

En el caso de las imágenes, una cámara de video casera no es lo óptimo ya que estas cámaras tienen baja resolución, dando como resultado imágenes con poca calidad. Para

capturar las imágenes de alta calidad, es mejor utilizar una cámara RGB, con una conexión a una tarjeta de entrada de video RGB a la computadora. Una alternativa, cuando se tienen las imágenes en disco duro o fotografías, es utilizar un dispositivo llamado scanner, el cuál puede registrar el color de las fotografías directamente dentro del formato dígital de la computadora. Los scanners de color proporcionan la mejor calidad de la imágen.

Además del equipo de entrada mecionado antes, se necesita equipo para convertir el audio y video analógico a datos digitales en la computadora. Esta capacidad es disponible vía tarjetas.

Se presenta una breve descripción de algunas de las tarjetas y software de manejo de video, utilizado para el desarrollo de las aplicaciones multimedia.

ActionMedia

ActionMedia de Intel Co. e IBM Co., proporciona la tecnología DVI, audio digital estéreo de alta calidad, video digital de toda la pantalla e imágenes de alta calidad completamente integrados con las otras cualidades de la PC en la que reside.

Contiene un coprocesador con memoria RAM separada, que permite que la descompresión y despliegue de video puedan operar en paralelo con el CPU.

El despliegue de video DVI está integrado al despligue VGA o XGA de forma que se puede superponer una imágen del plano DVI, sobre el plano VGA.

Tiene un coprocesador de audio que corre concurrentemente con el CPU. Este coprocesador es programable y es la interfaz a los canales de salida de audio. El software del sistema DVI que maneja al coprocesador, permite consigurarle varios parámetros de compresión y descompresión dando por resultados salidas monoaurales con calidad similar a radio AM, y estéreo con calidad similar a radio FM.

Contiene un módulo adicional de captura de audio y video, que permite hacer la entrada del aduio y video en tiempo real, la compresión y el guardado de los datos en disco duro.

DVA-4000

Esta tarjeta despliega video de alta calidad sobre computadoras IBM PC AT, compatibles IBM PS/2, Apple Macintosh II y Quadra. Proporciona manipulación en tiempo real de video con salida de alta resolución. Acepta entrada desde cualquier fuente VCR, cámaras de video ó reproductor de video laserdisc.

La tarjeta convierte el video de un formato analógico a digital, guardando cada uno de sus dos campos en memoria. Ahí cada uno de estos campos pueden modificarse, para después desplegarse como un solo video, o como videos diferentes. Para el despliegue los videos son convertidos nuevamente a análogos generando una señal idéntica a la producida por la tarjeta gráfica.

También permite capturar imágenes de video fijas en formato DIB.

MediaSpace

La tarjeta MediaSpace de VideoLogic proporciona compresión y descompresión de audio y video en tiempo real.

Utiliza las capacidades de despliegue de la tarjeta DVA-4000 para la auscencia de temblor en la pantalla.

Trabaja con los estandares internacionales de audio y video, JPEG y ADPCM y PCM. Puede comprimir, guardar y ejecutar secuencias de audio y video desde un amplio rango de dispositivos de almacenamiento magnético y óptico.

Captivator

Es una tarjeta de captura de video que trabaja con *"Microsoft Video for Windows"*. Permite grabar, editar y reproducir video, en formatos de 24, 16 y 8 bits. El video capturado puede ser guardado en formato AVI o YUV. El formato YUV compactado permite guardar secuencias de video con profundidad de color de 16-bit pero con un tamaño similar al de 8-bit.

Video para Windows

Es un software ampliamente utilizado para reproducción de video sin utilizar hardware especializado. Permite reproducción desde disco duro o CD-ROM, en máquinas de memoria limitada. Realizando compresión de cuadros, los cuales se encuentran intercalados junto con el audio en los archivos de video AVI.

2.3 Software

El software para multimedia debe proporcionar ejecución de audio y video, continúo y sin interrupciones o distorsiones, conjuntamente con las tarjetas que se utilicen en el sistema, y responder efectivamente a la interacción del usuario, dando un grado de flexibilidad que logre abarcar todas las diferentes gamas de presentación que son requeridas para cubrir un amplio rango de aplicaciones.

Para el propósito de esta discusión, dividiremos el tema de sloftware multimedia en tres partes:

1.- Sistema operativo es el software, que proporciona a las aplicaciones acceso a los recursos de la computadora.

2.- Software de autoría es el software que permite al autor crear programas de aplicación en una forma mas sencilla.

3.- Programación multimedia es el desarrollo de aplicaciones multimedia en lenguaje de programación.

2.3.1 Sistema operativo

El sistema operativo es el software que maneja los recursos de la computadora, donde se incluye un sistema de archivos que se encarga de manejar el disco flexible y almacenamiento en disco duro, el despliegue y demás tareas. El sistema operativo incluye una interface (API) que permite, al creador de una aplicación, tener acceso a los recursos del sistema desde su programa. La API es estandarizada así que permanece sin cambios no importando el hardware que está abajo.

Para proporcionar la independencia de hardware, los sistemas operativos son normalmente divididos en un kernel, el cual proporciona todas la funcionalidades básicas de la API, y un conjunto de manejadores, los cuáles casan el hardware específico al kernel. Así, para adaptar un sistema operativo al nuevo hardware, solo se necesita proporcionar los nuevos manejadores para este hardware. El kernel no tiene que ser cambiado.

Sin embargo, algunas veces el hardware nuevo incorpora herramientas que requieren cambios a la API. En este caso, es común proporcionar extensiones al sistema operativo. Una extensión es un módulo separado de software que modifica al API para acomodar las nuevas herramientas. Este debe tener acceso al nivel del manejador. Por ejemplo cuando el CD-ROM fue introducido, éste requirió de la adición de extensiones a los sistemas operativos existentes.

Los sistema operativos típicos para PC son MS-DOS y OS/2. Ambos son para un solo usuario; es decir, soportan sólo a un usuario interactuando con la máquina. Un sistema multiusuario tiene varios usuarios compartiendo una computadora; cada usuario tiene su propio teclado y terminal de despligue. Hay que hacer notar que el ampliamente usado Microsoft Windows no es técnicamente un sistema operativo; es un módulo de extensión que corre sobre MS-DOS.

El CPU no puede procesar datos de audio o video al mismo tiempo que los datos son leídos del dispositivo de almacenamiento. Varias estrategias de software han sido desarrolladas para atacar este problema en sistemas MS-DOS, pero la respuesta real es trabajar con un sistema operativo multitarea como OS/2. En OS/2, con programas de aplicación propiamente construidos, los recursos del sistema (CPU, bus, almacenamiento, etc.) nunca son ocupados completamente por una aplicación o una tarea dentro de una aplicación.

Las aplicaciones multimedia hacen grandes demandas al sistema operativo por el gran monto de datos que deberán ser manejados. Cuando el sistema de sonido necesita más datos de sonido, el sistema operativo debe mandar datos inmediatamente, o el sonido será interrumpido. Multimedia requiere que el sistema operativo proporcione estrategias para alimentar concurrentemente datos y ciclos de CPU a un número de actividades paralelas. Esta capacidad es llamada multitarea y no es disponible en muchos sistemas operativos. En un sistema monotarea, una aplicación multimedia debe poner su propia multitarea dentro de la aplicación.

La mejor propuesta es construir multitarea dentro de un sistema operativo que el mismo sea multitarea. OS/2 es un ejemplo de tal sistema. Éste maneja las necesidades de todas las aplicaciones, lo que hace posible correr multimedia a través de otras aplicaciones que pueden o no ser aplicaciones multimedia. Por ejemplo, bajo OS/2 se pueden correr programas de hojas de cálculo y elegir que ciertas celdas de la hoja de cálculo abran una nueva ventana y corran un video cuando hay un click por el usuario. Esto se puede hacer aún cuando la hoja de cálculo fue diseñada sin orientarse a multimedia, y el usuario puede interactuar con la hoja de cálculo mientras el video está corriendo.

La hoja de cálculo con video se realiza corriendo al mismo tiempo una aplicación que se encargue de recibir ordenes, para creara procesos multimedia (aplicación servidor multimedia). La forma en que se realiza la comunicación de los comandos, es a través de un protocolo de comunicación interprocesos, en el sistema operativo multitarea. En el caso anterior podría ser hecho vía el DDE (*Dynamic Data Exchange*), que es el protocolo para OS/2.

Multitarea es justamente el primer paso para implementar un sistema operativo para multimedia. Multimedia incorpora sus nuevas clases de recursos y debe ser posible que el sistema operativo pueda manejar. La necesidad de esto se muestra por los esfuerzos de la industria para proporcionar extensiones a los sistemas operativos existentes para ciertas necesidades multimedia. Por ejemplo, hay una extensión para MS-DOS llamda MSCDEX que le permite a este sistema operativo usar unidades de CD-ROM. Se necesita una extensión porque el CD-ROM es hardware que opera de forma diferente, pero debe poder ser accesado a través del sistema de archivos del sistema operativo en la misma forma que se accede al disco duro o flexible.

2.3.1.1 Microsoft Windows

Microsoft Windows 3.X, a pesar de no ser propiamente un sistema operativo, tiene características esenciales para el desarrollo de aplicaciones multimedia. como son: multitarea, comunicación entre procesos, interface gráfica (ventanas, botones,etc). Además de estas características, Windows proporciona servicios para el desarrollo de multimedia a través de sus extensiones.

Estas extensiones ofrecen al programador diversos servicios para controlar los dispositivos de los diferentes medios, permitiendo aparte de un completo control, la independencia de dispositivos, la cuál es una característica importante.

A continuación se veran,con más detalle, estas extensiones junto con el diseño de Multimedia Windows, que nos permitirá ver como se logra la funcionalidad que produce beneficios para la aplicación.

2.3.1.2 Servicios multimedia

Windows proporciona los siguientes servicios para multimedia:

- MCI*(Media Control Interface)*.
  - Interfaces basadas en cadenas o comandos para comunicación con los manejadores MCI de los dispositivos.
  - Manejadores de dispositivos MCI para reproducir y grabar audio waveform, reproducción de archivos MIDI y audio de disco compacto CD-ROM.
- Soporte API de bajo nivel para servicios relacionados con servicios multimedia.
  - Soporte de bajo nivel para reproducir y grabar audio con dispositivos de audio MIDI y waveform.
  - Soporte de bajo nivel para servicios de precisión del contador.
- Servicios de I/O Multimedia proporcionando *buffer* y *unbuffer* de archivos de I/O, y soporte para archivos RIFF *(Resource Interchange File Format)* de IBM/Microsoft.
- Un mapeador MIDI que soporta los servicios de composición MIDI estándar. Esto permite a los archivos MIDI ser creados independientemente de las actualizaciones del sintetizador MIDI del usuario final.
- Opciones en el Panel de Control que permite a los usuarios cambiar los manejadores de despliegue, actualizar el protector de pantalla, instalar manejadores de dispositivos multimedia, asignar sonidos en formato waveform a los sonidos de alerta del sistema, y configurar el mapeador MIDI.

- Funciones que llaman a interrupciones para los dispositivos del contador y MIDI, proporcionando respuesta en tiempo real.

Arquitectura de los servicios Multimedia Windows

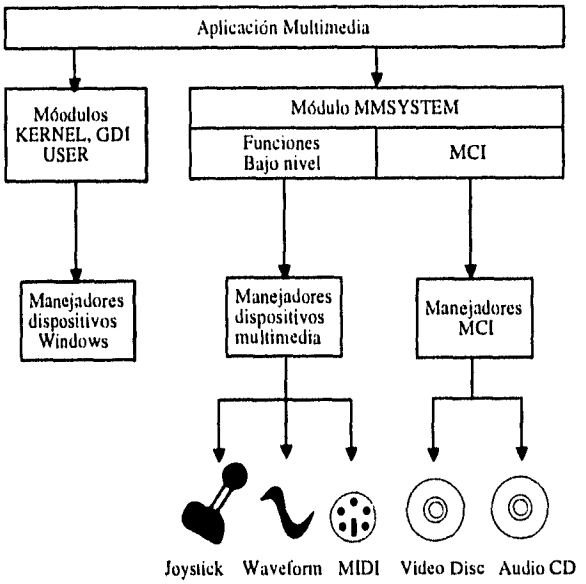
Aunque los servicios multimedia son proporcionados por un gran número de archivos, la arquitectura completa puede verse como si consistiera de unos pocos módulos de software:

- La biblioteca MMSYSTEM proporcionando los servicios de MCI *(Media Control Interface)* y el soporte de las funciones de bajo nivel.

- Manejadores de dispositivos multimedia que proporcionan comunicación entre las funciones de bajo nivel de MMSYSTEM y los dispositivos multimedia tales como waveform, MIDI y el hardware del contador.

- Los manejadores para el *Media Control Interface* que proporcionan control de alto nivel de los dispositivos de medios.

La siguiente imágen muestra la relación entre los módulos de Windows que proporcionan los servicios multimedia.



Relación entre Windows y las extensiones multimedia

Hay que hacer notar que las extensiones son justamente eso, extensiones. De esta forma, la arquitectura de Windows no es alterada, solamente aumentada. Multimedia se convierte en otra capa en el sistema. Cuando una aplicación requiere de servicios multimedia, la petición es pasada a MMSYSTEM, la cual es proporcinada a través de una biblioteca de ligado dinámico (MMSYSTEM.DLL) que es similar a las DLLs presentes en el ambiente de Windows.

Filosofía de diseño de Multimedia Windows

La arquitectura de los servicios multimedia esta diseñada alrededor de los conceptos de extensibilidad e independencia de dispositivos. La extensibilidad permite a la arqui-

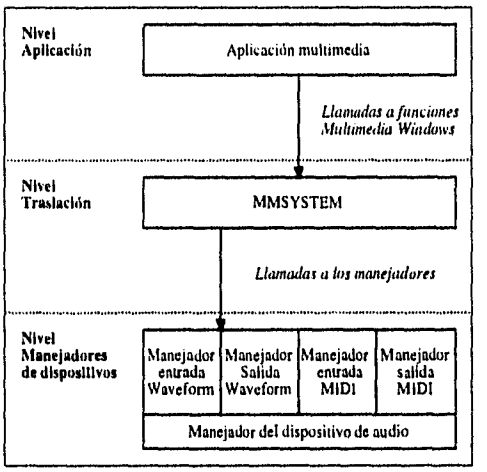
tectura del software, fácilmente, acomodar los avances tecnologicos sin modificarla. La independencia de dispositivos permite a las aplicaciones multimedia ser fácilmente desarrolladas para ejecutarse en un amplio rango de hardware proporcionando diferentes niveles de soporte multimedia. Tres elementos de diseño del sistema del software proporcionan la extensibilidad y la independencia de dispositivos:

- Una capa de traslación (MMSYSTEM) que aislan las aplicaciones de los manejadores de dispositivos y centralizan código independiente del dispositivo.

- Ligado en tiempo de ejecución que permite a la capa de traslación ligar los manejadores que necesita.

- Una interfaz de manejadores bien definida y consistente que minimiza código de casos especiales y hace más fácil la instalación y el proceso de actualización.

En la siguiente figura se muestra como la capa de traslación llevn una llamada a una función multimedia dentro de un manejador de dispositivo de audio.



Relación entre una aplicación y los manejadores de dispositivos

2.3.2 Sistemas de autoría

Existen una gran variedad de sistemas de autoría. Algunos de estos sistemas trabajan con un lenguaje de programación fácil y otros proponen la selección de los elementos de la aplicación por medio del mouse. Así, se debe seleccionar entre un sistema fácil que tenga limitaciones, o un lenguaje extremadamente flexible que requiere un esquema de programación para utilizar herramientas avanzadas.

Authorware

La estructura de la aplicación se crea seleccionando iconos y colocándolos en un cierto flujo. Sólo cuenta con 11 iconos, que se presentan en blanco y negro.

Se trabajan porciones pequeñas de una presentación, ya que sola permite una pantalla a la vez, sin opción de barra de desplazamiento. Pero proporciona la herramienta de mapeo de iconos que deja agrupar un conjunto de iconos, seleccionándolos y creando uno solo que los contenga. Esta forma de trabajo permite crear módulos que se pueden ocupar en varias aplicaciones.

Contiene iconos de despliegue de texto, gráficos, etc; e iconos de control de flujo, que son de fácil manejo. Además se pueden configuarar herramientas interactivas en las presentaciones. Por ejemplo, cuando se crea un ciclo para que el usuario responda, es posible limitar el número de intentos o el tamaño del tiempo para responder. Se ofrece preguntas de opción múltiple y herramientas que cazan si la respuesta es correcta . Manteniendo hisroria de intenios, aciertos y errores. Todas estas herramientas lo hacen un software apropiado para desarrollo de aplicaciones de adiestramiento.

Las producciones son compiladas y generan un sólo código ejecutable incluyendo el módulo runtime.

IconAuthor

IconAuthor de AimTech Corp; crea las aplicaciones en una forma visual a través de iconos.

Los iconos dan la forma a la presentación; ellos determinan la progresión de como los elementos fluirán. Una doble opresión de un botón sobre un icono revela también su ventana de contenido o su caja de diálogo. La caja de diálogo es donde se asigna detalles específicos, tales como el nombre del archivo de la imágen para ser cargada, el tamaño del tiempo del programa deberá de hacer pausas, o la clase de evento que puede activar la siguiente acción. En la mayoría de los casos, se llenan los blancos o se responde a las otras entradas de petición.

El programa ofrece siete grupos de folders para los iconos: flujo, entrada, salida, datos, multimedia, mejoras y extensiones.

IconAuthor tienen un buen soporte para el manejo de video,, lo cuá lo hace una buena opción si se va a trabajar con un reproductor de videodisc o videotape, junto con una tarjeta de sobreposición de video.

Los iconos están organizados dentro de grupos funcionales, los cuales están representados como folders en la barra de utilerías. Se tienen siete grupos de folder para los iconos de utilerías: flujo, entrada, salida, datos, multimedia, mejoras y extensiones. El folder de flujo incluye iconos para rutinas de control de flujo típicas, tales como menús, condicionales "if then", ciclos. Los iconos de entrada permite diseñar una aplicación que acomoda las entradas del usuario desde el teclado o ratón, mientras los iconos de salida manejan cosas, como poner imágenes situadas en la pantalla a archivos de gráficos o mandarlas a impresoras. Los folders de datos contienen iconos para manejar variables -por ejemplo, leer y escribir valores guardados en archivos de disco - y trabajar con archivos de dBase. El sistema del programa de variables hace esto fácil para seguir las respuestas del usuario

para anotaciones u otro tipo de retroalimentación.

Se puede usar la herramienta de alejamientos para moverse dentro y afuera del diagrama de flujo, y barras de desplazamiento que aparecen si el diagrama es muy largo o ancho para caber en la pantalla. Se permite abrir más de un diagrama de producción al mismo tiempo. . Permite mostrar en forma de mosaico o cascada todas las ventanas con diagramas abiertos.

IconAuhor además utiliza un sistema de manejo de archivos que forza a colocar todos los archivos de un sólo tipo en el mismo directorio. A pesar que la interfaz es fácil de usar, no llega a ser lo suficientemente amplia para producciones complejas.

AimTech ofrece un módulo de runtime, por el cuál se debe de pagar y permite distribuir las aplicaciones sin tener que proporcionar una copia total de IconAuthor para cada computadora que lo reproduzca.

IconAuthor tiene uno de los mejores soportes para el video. Si se planea utilizar videodisc o videotape en las producciones, esta es una opción razonable.

Este software ofrece una soporte de video flexible y poderoso, así que se ocupa en aplicaciones donde el video es muy importante.

ToolBook

ToolBook es fundamentalmente un ambiente de programación de alto nivel. Los conceptos básicos son: se dibujan varios objetos en la pantalla, entonces se crean los *scripts* debajo de ellos, que pueden causar uno o más resultados cuando un objeto dado es seleccionando. Estos scripts son pequeños programas escritos en OpenScript, que es el lenguaje de programación de ToolBook;

Dado que las producciones requieren más de una sola pantalla, Multimedia ToolBook,

necesita otro nivel de organización: este utiliza una metáfora de libro. Cada pantalla se describe como una página; todas las páginas en una producción son llamadas un libro. Dentro de cada página, se pueden tener objetos en diferentes niveles, los cuales están divididos además en el primer y segundo plano. La capa del segundo plano son útiles cuando se quiere crear una serie de páginas que compartan elementos comúnes, tales como una imágen o los botones de comandos tales como el de siguiente o salida.

Para crear un boton, simplemente se selecciona la utilería del botón desde la caja de utilerías y entonces se dibuja este sobre la pantalla. Después se edita las propiedades del objeto, lo cual incluye la clase de boton (tridimesional, sombreado, radio, o caja de contraseña) y el texto que será utilizado para etiquetarlo. Además se puede crear un script para el objeto botón.

Regularmente, se quiere que algo pase cuando el usario oprime el botón,así que se usa el manejador ButtonUp para disparar el script. Los manejadores son centrales para Multimedia ToolBook, ya ellos manejan mensajes desde los objetos y causan que el script se ejecute. Por ejemplo, el siguiente script podría causar que el programa avance a la siguiente página en la producción cuando el usuario oprime un botón:

```
to handle ButtonUp
    send next
end ButtonUp
```

ToolBook requiere que se haga algo de código desde el principio para crear aplicaciones simples. Existen herramientas de ayuda, tales como un grabador de script que funciona un poco como un grabador de macros en otros programas.

El acceso a las herramientas de Multimedia Windows por hacer llamadas a las funciones de MCI, pero la sintáxis y secuencias no siempre son intuitivas.

Multimedia ToolBook es excepcionalmente bueno en tipos de hiper-ligado de tareas gracias a herramientas especiales tales como sus habilidades para crear palabras sensibles. Con estas palabras se puede seleccionar una palabra o frase en el texto de una página y

asignar un script a éste. Esto significa que se puede accionar explicación de términos o ramificar a otras páginas, los fundamentos requeridos para crear una aplicación de hipertexto.

Una vez que se ha desarrollado la producción, fácilmente se pueden hacer copias y distribuirlas. Asymetrix incluye un paquete de runtime Windows, el cuál puede incluirse con las producciones sin pagar adicionalmente. Además se tiene la ventaja de poder aplicar una contraseña de protección a la producción en dos niveles: Un nivel limita el acceso para ver la producción, mientras el otro limita el acceso a la edición. Ninguno de los otros programas antes visto cosntruye herramientas de seguridad.

2.3.3 Programas de aplicación

Una de la formas de desarrollar las aplicaciones multimedia, es utilizando un lenguaje de autoría, que son ambientes de desarrollo de alto nivel, los cuales frecuentemente producen software limitado, ya que pueden haber facilidades adicionales que resultan ser difíciles o imposibles de realizar, como ejemplo podemos mencionar, la interacción directa con el video en una aplicación realizado con paquetes de autoría.

En estos casos se debe programar la aplicación desde lenguaje C++. La extensibilidad y potencia de C++, permite mejorar elementos que se plantean en multimedia logrando además una mayor versatilidad en el manejo de los dispositivos del hardware, ya que no se limitan las funciones que acceden a los dispositivos y el manejo de eventos puede hacerse directamente.

Para desarrollar las aplicaciones de esta forma se debe estar familiarizado con la programación en el ambiente Windows, con el objetivo de aprovechar, las extensiones de que se disponen, así como de los elementos gráficos y el manejo de multitarea.

Cuando se utilizan los servicios de multimedia, se deben incluir el archivo de encabezado MMSYSTEM.H en todo módulo fuente que llame a una función multimedia. Este archivo de encabezado depende de las declaraciones hechas en el archivo de encabezado

WINDOWS.H, así que primero se debe incluir WINDOWS.H. El MMSYSTEM.H proporciona prototipos de funciones, así como definiciones de tipos de datos y constantes.

Además de las bibliotecas normales de Windows, se deben ligar a la biblioteca MMSYSTEM.LIB.

Las aplicaciones generadas pueden existir en archivos ejecutables simples, o pueden consistir de un archivo de comandos junto con un módulo runtime o server. En la mayoría de los casos, los medios (audio,video, e imágenes) requeridos por la aplicación, deberán estar en archivos separados de los archivos ejecutables o de comandos. La programación de los medios desde un programa en C, se mostrará más adelante.

Capítulo 3

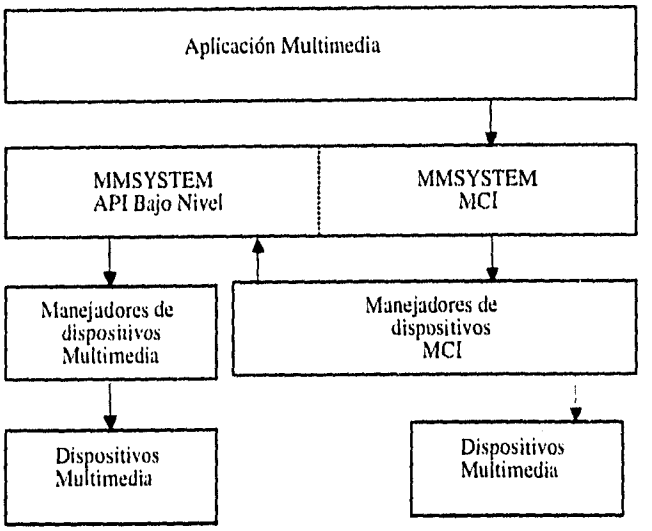
Media Control Interface (MCI)

Windows proporciona diferentes formas de programación de los medios de video, animación, audio y secuenciadores. Entre ellas se encuentra la programación haciendo llamadas a MCI *(Media Control Interface)*, que es una interfaz expandible que permite controlar virtualmente cualquier tipo de dispositivos brindando independencia de los mismos.

Las características de independencia de dispositivos y programación genérica que proporciona MCI son tan importantes que se recomienda el uso de esta interfaz para la programación en lugar de invocar funciones de bajo nivel y otras que proporciona Windows. Es por esta razón que en este capítulo explico de forma amplia las características de esta interfaz, dejando los detalles de la programación para el siguiente capítulo, con los propósitos de entender la forma en que se lleva a cabo la programación, y comprender el diseño y desarrollo de la biblioteca de objetos que presentaré en el capítulo 5.

3.1 La arquitectura MCI

Para proporcionar expansibilidad, MCI esta diseñado con una arquitectura que utiliza manejadores de dispositivos especiales para interpretar y ejecutar sus comandos.



Relación entre MCI y sus manejadores

Los manejadores MCI pueden controlar el hardware de los medios directamente o a través del API de bajo nivel. Los dispositivos más utilizados tales como los dispositivos de audio waveform y MIDI, son controlados a través de este API. Los dispositivos no soportados por el API de bajo nivel son controlados directamente, casi siempre a través del puerto serie.

3.2 Las interfaces de programación MCI

MCI proporciona dos interfaces de programación: una interfaz basada en comando-cadena y una interfaz comando-mensaje.

3.2.1 La interfaz comando-cadena

La interfaz de más alto nivel, y tal vez la más elegante, es la de comando-cadena,que permite pasar cadenas de texto al intérprete de comandos, y éste a su vez hace las llamadas reales a MCI. Esta interfaz permite utilizar los comandos en idioma inglés para comunicarse con dispositivos MCI. Por ejemplo, la siguiente cadena de comando ejecuta el archivo WAVE llamado "timpani.WAV" :

```
play timpani.wav
```

La interfaz comando-cadena está diseñada para ser usada con programación de alto nivel, sistemas de autoría tales como Authoware y Asymetrix ToolBook u otras herramientas como Visual Basic; pero representa una severa pérdida de eficiencia para una aplicación en lenguaje C++, ya que desperdicia mucho tiempo y recursos del sistema, cuando analiza e interpreta cada cadena antes de que pueda ser pasada a MCI.

3.2.2 La interfaz comando-mensaje

Esta interfaz usa un paradigma de paso de mensajes para comunicarse con los dispositivos MCI. Por ejemplo, el siguiente fragmento de código desarrolla la misma operación que el ejemplo anterior de comando-cadena:

```
mciSendCommand(wDeviceID,                          \*Identificador del dispositivo*\
               MCI_PLAY,                           \*Mensaje de comando*\
               0,                                  \*Banderas*\
               (DWORD)(LPVOID)&mciPlayParms);\*Bloque de parámetros*\
```

La interfaz comando-mensaje esta diseñada para ser usada en aplicaciones que requieran la interfaz de lenguaje C++ para controlar los dispositivos multimedia.

3.3 Conjunto de comandos de MCI

El grupo de comandos MCI está diseñado para proporcionar un conjunto genérico de instrucciones para controlar diferentes tipos de dispositivos de medios. Utiliza el mismo comando para empezar a reproducir un archivo de audio waveform, una pista de un videodisc o una secuencia de animación. Algunos tipos de dispositivos tienen capacidades únicas, tales como la capacidad del reproductor de una animación par utilizar un formato de tiempo basado en cuadros. Para estos dispositivos MCI proporciona comandos extendidos que son únicos para un tipo particular de dispositivo.

Los comandos de la interfaz comando-cadena proporcionan una buena visión del conjunto de comandos MCI. Cada uno de estos comandos es representado por un comando similar en la interfaz comando-mensaje. Por ejemplo, el comando equivalente para *close* es el mensaje MCI_CLOSE.

La siguiente tabla es una lista de algunos de los comandos más utilizados.

Comando	Acción
capability	Información acerca de las capacidades del medio
close	Cierra un dispositivo
info	Información tal como descripción del hardware asociado
open	Abre e inicializa un dispositivo para su uso
pause	Detiene la ejecución o grabación de un medio
play	Empieza la ejecución de un medio
record	Empieza la grabación sobre algún medio
resume	Concluye la ejecución o grabación de un medio
seek	Cambia la posición actual de un medio
set	Cambia los parámetros del dispositivo tales como formato de tiempo
status	Obtiene información acerca del estado del medio, como si se está ejecutando, o está en pausa
stop	Detiene ejecución o grabación de un dispositivo

Para una referencia completa de estos comandos véase *Multimedia Programmer's Reference* [4].

3.4 Clasificación de los dispositivos MCI

Los manejadores de los dispositivos MCI pueden ser clasificados como simples o compuestos. Los dispositivos simples no requieren un archivo de datos para la reproducción. Los reproductores de videodisc y reproductores de audio CD son ejemplos de dispositivos simples. Los dispositivos compuestos requieren un archivo de datos para la reproducción. Los reproductores de MIDI y audio de waveform son ejemplos de dispositivos compuestos.

Los archivos de datos asociados con un dispositivo compuesto se conoce como elemento del dispositivo. Ejemplos de elementos de dispositivo serían :

c:\sonidos\sonata.mid (MIDI)
c:\sonidos\campana.wav (WAVE)

3.5 Tipos de dispositivos estándar MCI

Un tipo identifica una clase de dispositivos MCI que responde a un conjunto de comandos comúnes. La siguiente tabla lista los tipos de dispositivos actualmente definidos.

Tipo de dispositivo	Descripción
animation	Dispositivo de animación
cdaudio	Reproductor de audio de CD
dat	Reproductor de cinta de audio digital
digitalvideo	Video digital en una ventana
other	Dispositivo MCI no definido
overlay	Dispositivo overlay (video analógico en una ventana)
scanner	Digitalizador de imágenes
sequencer	Secuenciador MIDI
vcr	Reproductor o grabador de video
videodisc	Reproductor de videodisc
waveaudio	Dispositivo de audio que ejecuta archivos de audio digitalizado waveform

3.6 Nombres de dispositivos

Cualquier tipo de dispositivo puede tener diferentes manejadores que comparten el conjunto de comandos pero operan sobre diferentes formatos de datos. Por ejemplo, hay diferentes manejadores para dispostivos de animación que utilizan el mismo conjunto de comandos pero los formatos de los archivos son diferentes. Para identificar de forma única un manejador MCI, se utilizan los nombres de dispositivos.

Los nombres de dispositivos son identificados en la sección de [mci] en el archivo SYSTEM.INI. Esta sección identifica todos los manejadores de dispositivos MCI para Windows. Un ejemplo de una sección [mci] sería:

```
[mci]
waveaudio=mciwave.drv
sequencer=mciseq.drv
MMMovie=mcimmp.drv
cdaudio=mcicda.drv
```

El nombre clave (la parte de lado izquierdo del signo igual) es el nombre del dispositivo. El valor correspondiente al nombre clave(parte derecha del signo) identifica el nombre del archivo del manejador MCI. Frecuentemente, el nombre del dispositivo es el mismo que el del tipo del dispositivo para el manejador, como es el caso para los dispositivos de waveform, sequencer y audio en el ejemplo anterior. El dispositivo "MMMovie" es un dispositivo de animación, pero usa un nombre único de dispositivo.

Si un manejador MCI es instalado utilizando un nombre de dispositivo que ya existe en la sección [mci], Windows agrega un entero al nombre del dispositivo del nuevo manejador, creando un nombre único de dispositivo. En el ejemplo, anterior un manejador instalado usando un nombre de dispositivo cdaudio podrá ser asignado a un nombre "cdaudio1". Un subsecuente dispositivo cdaudio podrá asignursele el nombre "cdaudio2".

3.7 Utilizando la interfaz comando-cadena

Para trabajar esta interfaz dentro de un programa en C++, se cuenta con la función:

mciSendString (*lpstrCommand,lpstrRtnString, wRtnString, wRtnLength,hCallback*)

donde:

lpstrCommand apunta a una cadena conteniendo la cadena de comandos con la siguiente forma:

comando nombre_del dispositivo argumentos

lpstrRtnString apunta a un buffer proporcionado por la aplicación para una cadena de regreso. Es utilizado en caso de que la función regrese algún valor, el cual es convertido a cadena.

wRtnString especifica el tamaño del buffer anterior.

hCallback es un manejador a una ventana que recibe el mensaje de MM_MCINOTIFY. Ésta es ignorada si el comando no contiene la bandera de notify.

Por ejemplo, para abrir ejecutar y cerrar un archivo MIDI, podría hacerse con las siguientes instrucciones:

```
mciSendString("open c:\ sonidos \ cancion.mid type midi alias cancion", NULL, 0,0L);
mciSendString("play cancion from 0", NULL, 0, 0L);
mciSendString("close cancion", NULL, 0, 0L);
```

Para una referencia completa de la interfaz comando-cadena, véase *Multimedia Programer's Guide*[5].

3.8 Utilizando la interfaz comando-mensaje

Una característica importante de MCI, que facilita la programación, es que no requiere de demasiadas funciones. Para la comunicación con el dispositivo se hace a través siempre de la función mciSendCommand, utilizando diversos comandos. De hecho existen una extensa variedad de comandos y banderas, para manejar los diversos dispositivos, teniendo la ventaja de que algunos son los mismos, sin importar el dispositivo que se está trabajando.

Los comandos se envían con la función

DWORD **mciSendCommand**(*wDeviceID, wMessage, dwParam1, dwParam2*)

donde:

wDeviceID es un **UINT** que identifica el dispositivo MCI que recibe el mensaje. Se utiliza el ID que se regresa cuando el dispositivo fue abierto.

wMessage es un **UINT** que especifica el mensaje. Los mensajes están definidos en el archivo MMSYSTEM.H.

dwParam1 es un **DWORD** que especifica banderas para el comando que indican opciones para el comando o campos válidos en el bloque de parámetros.

dwParam2 es un **DWORD** que se refiere al apuntador al bloque de parámetros para el comando. Si no se utiliza bloque , puede ser NULL.

Esta función regresa cero si no hay error. Si la función falla la palabra de más bajo

orden del valor regresado contiene un codigo de error, que se puede pasar a la función **mciGetErrorString** para obtener la descripción de este.

3.8.1 Mensajes de comandos

Los mensajes de comandos son los mensajes utilizados en la interfaz comando-mensaje donde cada mensaje consiste de los siguientes 3 elementos:

- Un valor constante que identifica al mensaje
- Un bloque de parámetros que especifica valores adicionales para el comando.
- Un conjunto de banderas que especifica opciones o campos de validación en el bloque de parámetros.

Por ejemplo, si consideramos el mensaje MCI_PLAY, tiene asociado la estructura MCI_OPEN_PARMS, y las banderas que se pueden utilizar pueden ser: MCI_WAIT, MCI_NOTIFY, que son banderas para todos los dispositivos y que indican la primera que espere hasta completar el comando antes de regresar el control al programa, y la segunda que avise cuando el comando se ha terminado de ejecutar. MCI_FROM y MCI_TO avisan que en el bloque MCI_PLAY_PARAMS los campos dwFrom y dwTo contienen información válida, indicando los límites para la ejecución del comando.

MCI define cuatro clasificaciones de comandos. Los comandos y opciones que comprenden las dos siguientes clasificaciones son definidas como el conjunto de comandos mínimo para cualquier manejador MCI:

Comandos del sistema son los manejados directamente por MCI más que por el manejador (MCI_BREAK, MCI_INFO).

Comandos requeridos son realizados por el manejador. Todos los manejadores de-

berían de soportar estos comandos y sus opciones(MCI_CLOSE,MCI_OPEN, etc).

Los comandos que comprenden las dos siguientes clasificaciones no son soportadas por todos los manejadores:

Comandos básicos, o comandos opcionales, son utilizados por algunos dispositivos. Si el dispositivo soporta un comando básico, entonces debe aceptar el conjunto de opciones para ese comando (MCI_PLAY, MCI_STOP ,etc.)

Comandos extendidos son adicionales para cierto tipo de dispositivos o manejadores. Estos incluyen nuevos comandos(tales como MCI_PUT y MCI_WHERE para el dispositivo de animación) y extensiones a comandos existentes (tal como la adición al comando de animación MCI_STATUS. MCI_ANIM_GETDEVCAPS_CAN_STRECH).

Si se necesita usar un comando básico o extendido, o alguna de sus opciones, se debe preguntar al manejador antes de tratar de utilizarlo.

En el *apéndice A* se muestra un sumario de los comandos antes mencionados.

Todos los mensajes de comandos tienen asociado un bloque de parámetros, y para ciertos dispositivos pueden existir bloques de parámetros extendidos. Por ejemplo MCI_OPEN tiene la estructura de datos MCI_OPEN_PARMS. Pero dispositivos tales como waveform, animation y overlay cuentan con bloques de parámetros adicionales, tal es el caso del bloque para overlay MCI_OVLY_OPEN_PARMS. A menos que se necesiten utilizar estos parámetros adicionales es suficiente utilizar la estructura MCI_OPEN_PARMS.

Ejemplos del uso de esta interfaz la veremos con el manejo de audio y animación de las secciones posteriores, donde se muestra el uso de algunos comandos, opciones y banderas más utilizadas. Si se requiere más información en *Multimedia Programer's Reference* [4] se encuentra la especificación de todos los mensajes, estructuras y opciones asociados a cada uno.

Capítulo 4

Programación de medios en ambiente windows

4.1 Objetos Multimedia

Los objetos multimedia que pueden manejarse bajo Windows pueden clasificarse de objetos activos y pasivos.

Los objetos activos son aquellos con los cuales puede uno interactuar y cambiar su estado de despliegue, en cambio, los pasivos solo son desplegados.

Los objetos activos están ligados al MCI, y son el video, el sonido y la animación , en cambio los objetos pasivos son las imágenes fijas o el texto, y no están asociados al MCI pero se necesita saber como desplegar una imágen de tipo bitmap.

A continuación presentaré las diferentes opciones con que se pueden programar los objetos multimedia.

4.2 Objetos Activos

4.2.1 Sonido

4.2.1.1 WAVE

Los archivos WAVE son piezas de sonido muestreado y almacenado como datos, que son guardados como archivos individuales en un disco o como elementos del archivo de recursos de una aplicación. Bajo windows tienen la extensión .WAV.

Con el hardware adecuado,y bajo el control de la computadora, un archivo de este tipo puede reproducir sonido con diversasa calidades desde una parecida a la de un telefóno, hasta la de un disco compacto, en monoaural o estereo.

Existen cuatro tipos de funciones bajo windows que permiten reproducir archivos de tipo WAVE. Cada uno se encuentra en un nivel de programación inferior al anterior.

Función Message Beep

Bajo circunstancias normales causa solo un sonido de beep sobre la bocina de la PC. Sin embargo esta tiene un argumento que puede realizar más cosas.

De hecho para las aplicaciones normales de Windows -sin las extensiones multimedia instaladas- el argumento para MessageBeep es ignorado. Sin embargo si se tienen una

tarjeta de sonido y los manejadores de los dispositivos de Windows instalados, se pueden utilizar diferentes argumentos para MessageBeep. Los argumentos estan definidos como constantes en el archivo MMSYTEM.H. Y son los siguientes:

MB_OK	Ejecuta el sonido SystemDefault del sistema
MB_ICONASTERISK	Ejecuta el sonido del SystemAsterisk
MB_ICONEXCLAMATION	Ejecuta el sonido del SystemExclamation
MB_ICONHAND	Ejecuta el sonido SystemHand
MB_ICONSQUESTION	Ejecuta el sonido SystemQuestion

Los sonidos que MessageBeep ejecuta en respuesta de estos argumentos están definidos en la sección de [sounds] del archivo WIN.INI. La sección de este archivo se ve como:

```
[sounds]
SystemAsterisk=chord.wav,Asterisk
SystemHand=chord.wav,Critical Stop
SystemDefault=ding.wav, Default Beep
SystemExclamation=chord.wav,Exclamation
SystemQuestion=chord.wav,Question
SystemExit=chimes.wav,Windows Exit
SystemStart=tada.wav,Windows Start
```

Este significa que se puede utilizar el siguiente comando:

```
MessageBeep(MB_ICONEXCLAMATION)
```

y el archivo chord.wav deberá ejecutarse. La sección de [sounds] del archivo WIN.INI es utilizada por Windows para definir sonidos y ligarlos a ciertos eventos del sistema, como por ejemplo en la aparición de mensajes de error, o en el momento de inicialización y salida del sistema. Se pueden definir estos sonidos y como serán utilizados en la parte de sonidos del panel de control de Windows.

Los archivos definidos inicialmente en la sección de [sounds] del WIN.INI son los proporcionados por Windows. Se pueden cambiar si se quiere pero hay que tener en cuenta que los archivos deben de ser pequeños.

Si se intenta correr un archivo wav que no existe, Windows intenta ejecutar el SystemDefault en su lugar. Además de generar el beep de protesta. Estos archivos de usos interno se encuentra en el directorio de Windows.

Existen otros dos posibles argumentos para MessageBeep. Si se pasa un valor de cero, se ejecutará el sonido de SystemDefault o el beep de la bocina si no encuentra el archivo WAVE. Si se le da el parámetro de -1 , ignora la seccion de [sounds] y solamente hace un beep.

Aunque se puede trabajar los archivos WAV de esta forma, no se pueden manejar las trayectorias de los archivos dentro de la aplicación misma.

Función SndPlaySound

Es la función más elemental para ejecutar archivos WAVE. Esta ofrece pocas opciones, y es fácil de trabajar. Acepta las rutas de los archivos o un apuntador a un archivo WAVE, que se mantenga en memoria, dejando al programador el problema de descifrarlo para ejecutarlo. Hay algunas limitaciones para esta forma de trabajar los archivos WAVE, pero es una buena forma si la aplicación realiza operaciones modestas con estos.

Hay dos argumentos para sndPlaySound: un apuntador y un conjunto de banderas. Un ejemplo de su uso sería:

```
sndPlaySound((LPSTR)"REDALERT.WAV",SND_ASYNC);
```

Esta llamada leerá un archivo nombrado REDALERT desde el disco duro y tratará de ejecutarlo.

La función regresa un valor de cierto si se ha ejecutado el archivo y falso en cualquier otro caso.

Además del nombre del archivo para sndPlaySound, se le puede pasar una llave de sonido, que es una cadena que se encuentra en la sección de [sounds] del WIN.INI. Por ejemplo si se pasa la cadena "SystemStart" como su primer argumento, este tratará de ejecutar el sonido asociado con la línea de SystemStart en WIN.INI, por default ésta es TADA.WAV.

El segundo argumento para sndSoundPlay ofrece controles. Se puede utilizar un operador OR para poner varias banderas.

Las opciones para estas banderas son las siguientes:

SND_ASYNC es la más utilizada le dice a la función que regrese inmediatamente al programa que la llamó tan pronto como se haya inicializado la ejecución.

SND_LOOP ejecuta el archivo específicado una vez tras otra , hasta que sea interrumpida explícitamente.

SND_MEMORY indica que el primer parámetro apunta al archivo cargado en memoria, en lugar de un archivo de disco.

SND_NODEFAULT si no se quiere que se ejecute el archivo SystemDefault en caso de no encontrar el archivo especificado.

SND_NOSTOP por default cuando se invoca al sndSoundPlay este termina la ejecución del archivo que se está ejecutando y comienza el nuevo. Esta bandera deshabilita esta forma de trabajar.

Dado que los archivos WAVE son regularmente largos, se cargan en la memoria global.

Hay pocas cosas que no se pueden hacer con la función sndSoundPlay. Se puede

averiguar si un archivo se está ejecutando; utilizar archivos de disco, en memoria y como recursos. Pero hay una pequeña razón para utilizar algo más elaborado. De hecho es la mayor limitación para utilizar sndPlaySound. Sólo trabaja con archivos WAVE pequeños. La documentación de Micrososft sugiere que el límite sea alrededor de 100 Kb, un poco más grande si se tiene memoria suficiente.

Se puede utilizar sndPlaySound para ejecutar efectos de sonido cortos en el software, pero no sería la manera adecuada para desarrollos largos. Además su estructura no permite en sí misma, escribir aplicaciones que utilicen muchos archivos de sonidos.

La forma correcta para crear una aplicación que quiera concatenar una serie de archivos WAVE es tener la función que ejecuta el archivo y le dice a la aplicación cuando ya está lista para ejecutar otro. Lo cuál se puede simular con sndPlaySound, pero no es eficiente.

Esta es una facilidad que sndSoundPlay no ofrece y es una de las razones para ejecutar los sonidos bajo las extensiones multimedia Windows.

Llamadas al MCI

En la mayoría de las aplicaciones que utilizan archivos WAVE, se puede obtener el mejor balance de funcionalidad y complejidad de código al utilizar las llamadas a MCI.

Un fragmento de código que abre, ejecuta y cierra un archivo WAVE, a través de cadenas MCI, sería el siguiente:

```
mciSendString("open waveaudio",NULL,0,0L);
mciSendString("play waveaudio to 500",NULL,0,0L);
mciSendString("close waveaudio",NULL,0,0L);
```

Una mejor propuesta, para el manejo de archivos WAVE, es hacer las llamadas a las

funciones MCI directamente.

Para utilizar un reproductor de archivos WAVE , se debe abrir el dispositivo de sonidos en el sistema como un dispositivo MCI, referirse al archivo WAV que se quiere ejecutar, y entonces ejecutarlo. Al contrario de sndSoundPlay, la interface MCI mandará un mensaje a la ventana que se elija cuando se está ejecutando un sonido, de esta forma se puede seguir el desarrollo de la reproducción. Una de las cosas que se debe de hacer al terminar de ocupar el dispositivo WAVE es cerrar el manejador del sonido MCI abierto anteriormente.

Si se quieren ejecutar archivos de sonidos WAVE secuencialmente, se puede lograr manejando el mensaje que se genera con la conclusión de uno para iniciar el siguiente.

El primer paso es abrir el dispositivo MCI de sonido. El siguiente código muestra como se hace:

```
MCI_OPEN_PARMS mciopen;
DWORD rtrn;
char b[STRINGSIZE+1];
int id;

mciopen.lpstrDeviceType="waveaudio";
mciopen.lpstrElementName=path;
if((rtrn=mciSendCommand( 0,
                         MCI_OPEN,
                         MCI_OPEN_TYPE | MCI_OPEN_ELEMENT,
                         (DWORD)(LPVOID)&mciopen)) != 0L){
    mciGetErrorString(rtrn,(LPSTR)b,STRINGSIZE);
    DoMessage(hwnd,b);
    return(0);
}
id=mciopen.wDeviceID;
```

El primer argumento de mciSendCommand es un valor de dipositivo ID, si no se le

da un valor como en el ejemplo anterior, es que se va a abrir un dispositivo.

El segundo argumento le dice a mciSendCommand lo que va a hacer. En este caso para abrir un dispositivo su argumento debe ser MCI_OPEN. el tercer argumento es un conjunto de banderas, el cual le dice a la función que elementos de su cuarto argumento utilizará. Las banderas en este argumento tendrán diferentes aplicaciones en diferentes tipos de llamadas.

El cuarto argumento para mciSendCommand es un apuntador a una estructura de parámetros. Hay diferentes tipos de estructuras para utilizarse en esta llamada.

La información que mciSendCommand requiere para abrir el dispositivo es proporcionado por el objeto MCI_OPEN_PARMS pasado a éste. En este caso "waveaudio", y el nombre del archivo WAVE que se va a ejecutar con su trayectoria, que en este caso se encuentra en la variable *path*.

El valor que regresa la función es cero si todo resulta bien, o regresa un código que está definido en una lista de constantes numéricas, pero si no se quiere trabajar directamente con ella, existe una función : mciGetErrorString, que regresa una descripción del error que ocurrió.

Asumiendo que la llamada a mciSendCommand para abrir el dispositivo waveform se ejecutó correctamente, el valor en wDeviceId del objeto MCI_OPEN_PARMS debe contener el valor del identificador del dispositivo para el reproductor de waveform.

Habiendo abierto el dispositivo reproductor de wave, el siguiente paso es ejecutar el archivo:

```
MCI_PLAY_PARMS mciplay;
mciplay.dwCallbak=(DWORD)hwnd;

if((rtrn=mciSendCommand(id,
                        MCI_PLAY,
                        MCI_NOTIFY,
```

```
                    (DWORD)(LPVOID)&mciplay))!=0L){
        mciSendCommand(soundID,MCI_CLOSE,0,NULL);
        mciGetErrorString(rtrn,(LPSTR)b,STRINGSIZE);
        DoMessage(hwnd,b);
        return(0);
    }
```

La segunda línea asigna al campo dwCallbak de la estructura MCI_PLAY_PARMS que será pasado a la función mciSendCommand. Este valor es un manejador de ventana de tipo HWND. Y esto le dice a MCI donde mandar un mensaje indicando la finalización del sonido que se ha empezado a ejecutar, cuando termine de ejecutarse el archivo de sonido. La bandera MCI_NOTIFY en el tercer argumento para mciSendCommand le dice a MCI que el valor de dwCallbak es un manejador de ventana válido, y que lo use de acuerdo a esto.

Hay que tomar en cuenta, que si la llamada generó un error, se debe cerrar el dispositivo, llamando nuevamente la función mciSendCommand, con el mensaje MCI_CLOSE.

Habiendo ejecutado estas dos funciones, lo único que queda es responder al mensaje que la función manda a la ventana, que nos indica cuando el sonido ha finalizado. Se genera un mensaje MM_MCINOTIFY. Una forma de manejar el mensaje sería:

```
case MM_MCINOTIFY:
    mciSendCommand(LOWORD(lParam), MCI_CLOSE,MCI_WAIT, NULL);
    break;
```

Cuando se produce un mensaje MM_MCINOTIFY, la parte baja de la palabra del valor lParam pasado al manejador de mensajes contiene el ID del dispositivo que termino de ejecutarse.

Si se quisiera parar la ejecución de un sonido en algún momento, el código para hacerlo sería como el siguiente:

```
MCI_GENERIC_PARMS mcigen;

mcigen.dwCallback=hwnd;
mciSendCommand(id,
                MCI_STOP,
                MCI_NOTIFY|MCI_WAIT,
                (DWORD)(LPVOID)&mcigen);
```

Esta llamada a mciSendCommand parará la ejecución del archivo y además pide mandar una notificación, cuando la ejecución termine.

Llamadas a waveOut

Son las llamadas de más bajo nivel de funciones disponibles bajo las extensiones de multimedia, para ejecutar archivos WAVE.

Utilizando estas funciones, se puede controlar la ejecución de sonidos hasta los límites de la funcionalidad provista por el hardware de sonido. Sin embargo, utilizar estas funciones requiere más líneas de código.

La estructura de una función utilizando llamadas a waveOut es similar a las llamadas a MCI discutidas anteriormente. Éste ejecuta un sonido y señala a la ventana que se escogió cuando el archivo se ha finalizado. Sin embargo, utilizar waveOut requiere que el sonido sea cargado en la memoria, en lugar de pasar una ruta a un archivo de disco. Esto involucra trabajo con los engorrosos detalles de los archivos RIFF.

Utilizando llamadas a waveOut, se deben ajustar diversos parámetros del sonido que se va a ejecutar. Incluyendo volúmen, velocidad de reproducción, y tono. Hay diferentes parámetros que las tarjetas de sonido pueden soportar y a las cua'les las extensiones multimedia proporcionan acceso. Sin embargo, algunos de estos parámetros no son soportados por las tarjetas. De esta forma, se debe saber cuáles son las características de una tarjeta.

En la siguiente sección de código se ejecuta un archivo WAVE con llamadas a waveOut, se asume que el archivo ha sido previamente cargado en memoria desde un archivo RIFF, y actualmente se encuentra en un buffer manejado por wavehandle. Su formato relevante esta en PCMWAVEFORMAT, llamado waveformat.

Para empezar, hay que abrir el el dispositivo de ejecución de WAVE para utilizarlo con las llamadas waveOut:

```
HWAVEOUT hwaveout;
char b[STRINGSIZE+1];
if ((rtrn=waveOutOpen((LPHWAVEOUT)&hwaveout,
                      WAVE_MAPPER,
                      (LPWAVEFORMAT)&waveformat,
                      (LONG)hwnd,
                      0L,
                      CALLBACK_WINDOWS)) !=0){
    waveOutGetErrorText(rtrn,(LPSTR)b,STRINGSIZE);
    return(0);
}
```

La función waveOutOpen intenta abrir el dispositivo para ejecución del sonido. En aplicaciones sofisticadas con diferentes reproductores de sonido, se pude especificar cuál dispositivo abrir. El segundo argumento para waveOutOpen debe ser el identificador ID del dispositivo de sonido que se quiere abrir, o la constante WAVE_MAPPER, la cual le dice a Windows que encuentre una tarjeta de sonido disponible.

Además, en un sistema con múltiples reproductores de sonido se debe averiguar el ID del dispositivo obteniendo la información de cuántos dispositivos se tienen y revisando cada uno de ellos para ver cuál es el que cumple con las características con las que se está trabajando.

El número de dispositivos se obtiene como sigue:

```
int count;
count=waveOutGetNumDevices();
```

Sabiendo el número de dispositivos en el sistema, se puede pasar a través de cada uno por llamar waveOutGetDevCaps, el cuál regresa información acerca de un reproductor específico.

En teoría, es práctico seleccionar un dispositivo reproductor de WAVE basado en la información dada por waveOutGetDevCaps. en la práctica, probablemente se quiera mejor desplegar un menú, mostrando la lista de los reproductores de WAVE disponibles y permitir al usuario la selección de alguno.

El argumento waveformat pasado a la función waveOutOpen es un apuntador lejano a un objeto PCMWAVEOUT que ha sido previamente llenado con datos del archivo que será ejecutado. El argumento hwnd debe ser un manejador a una ventana a la cuál wave-OutOpen mandará los mensajes. La bandera CALLBACK_WINDOW le indica que ha sido pasado un manejador de ventanas para este propósito. Es posibles pasar una dirección de función, en tal caso la bandera deberá de ser CALLBACK_FUNCTION.

Si la llamada a waveOutOpen tiene éxito, llena el objeto apuntado por HWAVEOUT para su primer argumento con información acerca del dispositivo abierto. Este sirve como un manejador para las subsecuentes llamadas a waveOut.

Como las llamadas a MCI las funcioens waveOut regresan cero si todo esta bien o un código de error. La funció waveOutGetErrorText, regresa una descripción en texto de la condificón de error si una llamada a waveOut corre con dificultades. Aqui esta como se debe utilizar:

```
waveOutGetErrorText(rtrn,(LPSTR)b,STRINGSIZE);
```

El valor de regreso rtrn fue dado por una llamada a waveOut.

Una vez abierto el dispositivo para utilizar waveOut, es necesario preparar el encabezado wave que será utilizado en la ejecución del sonido. Hay una llamada para manejar éste, waveOutPrepareHeader. El encabezado es un objeto WAVEHDR. Por razones no explicadas en la documentación de Microsoft, waveOut deberá residir en memoria global, lo cual significa que debe ser localizado dinámicamente.

```
GLOBALHANDLE waveheader;
LPWAVEHDR pwaveheader;
if((waveheader=GlobalAlloc(GMEM_MOVEABLE|GMEM_SHARE,
                           (long)sizeof(WAVEHDR)))==NULL){
    DoMessage(hwnd,"Memory allocation error");
    return(0);
}
if((pwaveheader=(LPWAVEHDR)GlobalLock(waveheader))==NULL){
    GlobalFree(waveheader);
    DoMessage(hwnd,"Memory locking error");
    return(0);
}
```

En este punto se deben cerrar los datos WAV también:

```
if((wavepointer=(HPSTR)Globallock(wavehandle))==NULL){
    GlobalFree(waveheader);
    DoMessage(hwnd,"Memory locking error");
    return(0); }
```

El objeto WAVEHDR incluye un número de campos que se deben actualizar antes de llamar a waveOutPrepareHeader. Específicamente, se debe decir a waveOut donde encontar el archivo de datos WAVE y que tan grande es.

El tamaño de los datos serán el tamaño del pedazo regresado por mmioDescend cuando se desciende dentro de subpedazo del archivo WAVE para ser ejecutado.

Aqui está la llamada a waveOutPrepareHeader:

```
pwaveheader → lpData=(LPSTR)wavepointer;
pwaveheader → dwBufferLength=mmSub.cksize;
pwaveheader → dwFlags=0L;
pwaveheader → dwLoops=0L;
if((rtrn=waveOutPrepareHeader(hwaveout,pwaveheader,sizeof(WAVEHDR))) !=0 ) {
    waveOutUnprepareHeader(hwaveout,
    pwaveheader,sizeof(WAVEHDR));
    waveOutClose(hwaveout);
    waveOutGetErrorText(rtrn,(LPSTR)b,STRINGSIZE);
    DoMessage(hwnd,b);
    return(0);
}
```

Como en las llamadas MCI es importante cerrar un dispositivo waveOut en el evento si ha ocurrido un error. Además es importante llamar waveOutUnprepareHeader ya que este libera la memoria previamente obtenida por waveOutPrepareHeader.

Después de todos estos pasos waveOut está listo para hacer algún ruido. La función para poner a ejecutar un archivo WAVE es waveOutWrite. De hecho, se escribiran los datos WAVE al dispositivo WAVE. Como se hace en la siguiente llamada:

```
if((rtrn=waveOutWrite(hwaveout,pwaveheader,sizeof(WAVEHDR))) != 0 ){
    waveOutUnprepareHeader(hwaveout,pwaveheader,sizeof(WAVEHDR));
    waveOutClose(hwaveout);
    waveOutGetErrorText(rtrn,(LPSTR)b,STRINGSIZE);
    DoMessage(hwnd,b);
    return(0L);
}
```

Todos los objetos en la llamada a waveOutWrite sonarán familiares. El objeto hwave-

out es el manejador HWAVEOUT llenado por waveOutOpen. El objeto pwaveheader es el objeto WAVEHDR obtenido previamente y llenado por waveOutPrepareHeader. El tercer argumento le dice a waveWrite que tan grande es el segundo argumento. Si waveOutWrite regresa un cero, el sonido se está ejecutando. Como con las llamadas con MCI, el manejador waveOut manda un mensaje a la ventana que se indico en llamada a waveOutOpen cuando el sonido ha finalizado.

Cuando un sonido inicializado por una llamada a waveOut finaliza su ejecución, manda el mensaje MM_WOM_DONE a la ventana especificada de regreso en la llamada a waveOutOpen. Aqui se muestra como el manejador de mensajes podría verse:

```
case MM_WON_DONE:
    waveOutUnprepareHeader((HWAVEOUT)wParam,
                           (LPWAVEHDR)lParam,sizeof(WAVEHDR));
        waveOUtClose(HWAVEOUT)wParam);
    if(wavehandle != NULL){
        GlobalUnlock(wavehandle);
        GlobalFree(wavehandle);
    }
    if(waveheader != NULL){
        GlobalUnlock(waveheader);
        GlobalFree(wavehandle);
    }
    break;
}
```

El buffer global de memoria que guarda los datos WAVE que serán ejecutados por waveOut debe permanecer válido hasta que el sonido pare la ejecución . Esto significa que no se puede liberar éste hasta que un mensaje de MM_WOM_DONE es mandado. En la práctica, el camino más fácil es hacer el manejador como una variable global.

Como con las otras funciones de ejecución de sonido discutidos antes, se puede parar un sonido que se ha empezado a ejecutar por waveOut. La llamada waveOutReset parara un

sonido y mandará un mensjae de MM_WOM_DONE para liberar lo buffers. He aquícomo se usa:

```
waveOutReset(hwaveout);
```

El argumento hwaveout es el manejador HWAVEOUT que indica el archivo waveOut abierto.

Como se nota, una de las razones para usar las llamadas waveOut para manejar la reproducción de dispositivos WAV , es que se pueden utilizar waveOut para modificar diversos parámetros, tales como el volúmen. Actualizar el volúmen para un sonido usando una llamada a waveOut no es complicado. La función que hace esto se vería así:

```
waveOutSetVolume(id,volume);
```

El argumento id para waveOutSetVolume es el indefintificador del dispositivo originalmente pasado a waveOutOpen. Si se utiliza la constante WAVE_MAPPER, sin embargo, el dispositivo actual en uso no será bien reconocido. Hay otra función para tener cuidado de este problema, waveOutGetID - este deriva un identificador de dispositivo desde un objeto HWAVEOUT. Aquí esta como se vería:

```
WORD id;
waveOutGetID(hwaveout,(LPWORD)&id);
```

asumiendo que la función wave0utSetVolume ha regresado un cero, indicando que le gustó el sabor del argumento hwaveout que se le pasó, el argumento id contendrá un dispositivo válido ID. El argumento del volúmen para waveOutSetVolume es un entero largo sin signo. La palabra en su parte baja contiene el canal izquierdo de volúmen y la parte alta contiene el canal derecho. Si la tarjeta de sonido es monoaural , la parte baja de la palabra será el control del volúmen y la parte alta será ignorada.

Los valores de volúmen van desde cero para el silencio hasta 65535 para todo el sonido. Si se pone el argumento de sonido para waveOutSetVolume a 0xffffffffL, ambos canales se pondrán a la máxima potencia.

Hay algunas cosas que se deben tener en cuenta para utilizar el waveOutSetVolume. La primera es que no todas las tarjetas soportan ésta, y puede determinarse al usar la función waveOutGetCaps. Si no se soporta esta característica y se intenta hacer no pasa nada.

Los cambios de volúmen no se reestablecen solos cuando un sonido para o termina de ejecutarse. Si se cambia el volúmen en la mitad de un sonido este resentirá el cambio hasta que explícitamente se vuelva a cambiar.

4.2.1.2 CD-ROM

A diferencia de otros dispositivos el CD sólo admite comandos referidos a la reproducción y no a la captura de sonido. Por otro lado, un CD está compuesto por un número determinado de pistas, que a su vez se dividen de una forma lógica en minutos, segundos y cuadros. En teoría es posible posicionar la cabeza lectora en cualquier cuadro determinado, pero en la práctica existen oscilaciones en el sistema mecánico que limita la precisión.

Como es lógico, el término "pistas" está muchas veces lo asociamos directamente a lo que consideramos como canciones, aunque es irrelevante para efectos de programación. No existe un límite teórico de canciones o pistas por disco, aunque estas no deben sobrepasar 99 por cuestiones de referencia(normalmente se utilizan pocas, del orden de 10 o 20). Lo que si existe es un número máximo de duración del disco debido a limitaciones propias del medio que aproximadamente son 73 minutos.

Una pista determinada tiene una posición absoluta en el disco que está identificada por el tiempo de comienzo. La siguiente pista determina el final de la primera, pero sólo hasta cierto punto, puesto que pueden existir algunos segundos intermedios. Para determinar la longitud exacta de una pista debemos consultarlo con el comando apropiado.

No existe un modo único a la hora de considerar la organización de un disco. Así, el sistema de pistas:minutos:segundo es sólo la posibilidad más común, siendo igualmente interesante minutos:segundos:cuadros, en este último caso consideraremos al disco absolutamente temporal, sin organización de pistas. Debemos advertir que la primera canción en este caso no comienza en la posición 0:0:0, puesto que existe una tabla de índices, no accesible directamente, que no puede ser reproducida. Otras formas de representar el tiempo corresponden a los estándares de la SMPTE(Society of Motion Picture and Television Engineers) que, al igual que los anteriores, también se empaquetan en un número de tipo DWORD.

La comunicación entre los comandos que envía el programa y el driver que gestiona el CD se realiza mediante la función mciSendCommand() que tiene 4 parámetros: el identificador del dispositivo, el mensaje, indicadores sobre el mensaje y un apuntador a una estructura(bloque de parámetros).

El primer comando debe ser de apertura de dispositivo, o sea MCI_OPEN, para lo cual vamos a rellenar nuestra primera estructura que es MCI_OPEN_PARMS. En el campo lpstrDeviceType asignamos el nombre del periférico que nos interesa, que en este caso es "cdaudio".

Llamados a la función mciSendCommand con el primer parámetro como NULL, ya que en este momento aún no existe dispositivo asociado. El segundo parámetro es la bandera MCI_TYPE, que le indica , que en el cuarto parámetro está incluida la especificación del tipo de dispositivo para abrir.

Si no ocurre ningún error mciSendCommand regresa cero . En caso contrario, regresa el código de error. Si se quiere obtener el texto del error que se produjo, se utiliza la función MCI_GetErrorString.

Si se abrió el dispositivo, se asigna a id el identificador del dispositivo, que regresa mciSendCommand, en el campo wDeviceID de la estructura MCI_OPENPARMS.

```
MCI_OPEN_PARMS mciOpen;
int id;
DWORD rtrn;

mciOpen.lpstrDeviceType="cdaudio";
if((rtrn=mciSendCommand(NULL,
                        MCI_OPEN,
                        MCI_OPEN_TYPE,
                        (DWORD)(LPVOID)&mciOpen ))
    return(dwReturn);
}
id=mciOpen.wDeviceID;
```

Una vez abierto el dispositivo, ya se le pueden enviar comandos que actúen en función de las pistas o del tiempo.

La posición de la cabeza lectora, como hemos dicho con anterioridad, se representa en un valor DWORD que puede estar en diferentes formatos.

El MSF (minuto:segundo:cuadro) y el TMSF (pista:minuto:segundo :cuadro), son los formatos más utilizados. Para cambiar el formato se deben utilizar el comando MCI_SET, subcomando MCI_SET_TIME_FORMAT con la estructura MCI_SET_PARMS en el campo dwTimeFormat. Las banderas MCI_FORMAT_MSF,MCI_FORMAT_TMSF, MCI_FORMAT_MILLISECONDS, son los valores posibles. Para crear un valor en formato MSF se utiliza la macro MCI_MAKE_MSF(minutos, segundos,cuadros), así como para extraer la información estan las macros MCI_MSF_MINUTE, MCI_MSF_SECOND, MCI_MSF_FRAME. Para TMS ocurre algo parecido , con las macros MCI_MAKE_TMSF, MCI_MSF_TRACK.

El siguiente código asigna el formato del tiempo de TMSF, en caso de que ocurra algún error, se debe de cerrar el dispositivo:

```
MCI_SET_PARMS mciSet; /* Estructura asociada a MCI_SET */
mciSet.dwTimeFormat=MCI_FORMAT_TMSF;
if((rtrn=mciSendCommand(id,
                        MCI_SET,
                        MCI_SET_TIME_FORMAT,
                        (DWORD)(LPVOID)&mciSet)) !=0 ){
    mciSendCommand(id,MCI_CLOSE,0,NULL);
    return(0);
}
```

Lo siguiente que se necesita saber es el número de pistas del disco compacto en cuestión. Se manda llamar la función mciSendCommand con el mensaje MCI_STATUS, y nos regresa en el campo mciStat de MCI_STATUS_PARM , el número de tracks de último track en el disco. Las pistas empiezan en 1.

```
MCI_STATUS_PARMS mciStat;
unsigned int tracks;

mciStat.dwItem=MCI_STATUS_NUMBER_OF_TRACKS;
if((dwReturn=mciSendCommand(id,
                            MCI_STATUS,
                            MCI_STATUS_ITEM,
                            (DWORD)(LPVOID)&mciStat)) !=0 ) {
    mciSendCommand(id,MCI_CLOSE,0,NULL);
    return(0);
}
tracks=(unsigned int)mciStat.dwReturn;
```

Antes de poder ejecutar una pista necesitamos saber, cual es su longitud. Por ejemplo si quisieramos ejecutar la última pista (tracks).

```
MCI_STATUS_PARMS mciStat;
unsigned long time;

mciStat.dwItem=MCI_STATUS_LENGTH;
mciStat.dwTrack=tracks;
if((rtrn=mciSendCommand(id,
                        MCI_STATUS,
                        MCI_STATUS_ITEM | MCI_TRACK,
                        (DWORD)(LPVOID)&mciStat)) !=0 ) {
        return(0);
}
time=mciStat.dwReturn;
```

Con lo anterior ya se puede ejecutar la ultima pista del disco. El comando que inicia la reproducción es MCI_PLAY. Esta función admite los indicadores:MCI_FROM y MCI_TO que regulan la información contendida en la estructura MCI_PLAYS_PARMS. En este comando, al igual que en otros muchos tenemos dos formas de programación: una de ellas es directa que se refiere a la ejecución del comando sin mayor preámbulo, pero también se puede utilizar la bandera MCI_WAIT para recuperar el control sólo cuando el comando se haya ejecutado en su totalidad. Sin duda el paralizar la aplicación no interesa demasiado, lo que nos obliga a buscar otras alternativas de programación. Una solución es lanzar el comando y preguntar periódicamente la ejecución del mismo, o bien construir una función call-back y especificar la badera MCI_NOTIFY, con lo cual dicha función será llamda notificando la finalización del comando. El especificar el indicativo obliga a considerar el rampo dwFrom como comienzo de ejecución, ocurriendo otro tanto con la finalización y el indicativo MCI_TO junto con el campo dwTo.

```
MCI_PLAY_PARMS mciPlay;

mciPlay.dwFrom=MCI_MAKE_TMSF(track,
                             MCI_MSF_MINUTE(0),
                             MCI_MSF_SECOND(0),
                             MCI_MSF_FRAME(0));
mciPlay.dwTo=MCI_MAKE_TMSF(track,
```

```
                                    MCI_MSF_MINUTE(time),
                                    MCI_MSF_SECOND(time),
                                    MCI_MSF_FRAME(time));

    mciPlay.dwCallback=(DWORD)hwnd;
    if((rtrn=mciSendCommand(id,
                                    MCI_PLAY,
                                    MCI_FROM | MCI_TO | MCI_NOTIFY,
                                    (DWORD)(LPVOID)&mciPlay)) !=0 ){
        return(0); }
```

Como se pude observar, en el código anterior se le pasa como bandera MCI_NOTIFY, que manda un mensaje cuando termina la ejecución de la pista. Una forma de manejar ésta sería, mandar un mensaje para cerrar el dispositivo.

```
case MM_NOTIFY:
    mciSendCommand(LOWORD(lParam),MCI_CLOSE,MCI_WAIT, NULL);
    break;
```

Además de los comandos anteriores existen otros, que hacen referencia a un posicionamiento(SEEK) y detención(PAUSE/STOP). Recordando que cada comando suele tener una estructura asociada, así como la posibilidad de ejecución directa, o en diferido, con la correspondiente notificación (opciones de WAIT y NOTIFY).

4.2.2 Video

Para poder ejecutar archivos AVI se debe tener instalado el manejador en Windows. El manejador se llama MCIAVI.DRV y debe estar en \WINDOWS \ SYSTEM del disco duro. Además deberá de estar listado en la sección [MCI] del archivo SYSTEM.INI, con una línea como :

```
AVIVideo=mci.drv
```

El manejador es instalado automáticamente si se instala Video para Windows.

La mayoría de las funciones para AVI pueden ser manejadas a través de llamadas a la interfaz MCI. Por ejemplo un segmento de código para ejecutar un archivo AVI es:

```
DWORD PlayFlick(LPSTR path, HWND hwnd) {
    MCI_DGV_OPEN_PARMS mciopen;
    MCI_DGV_PLAY_PARMS mciplay;
    MCI_GENERIC_PARMS mcigen;
    DWORD rtrn;
    char b[STRINGSIZE+1];
    mciopen.lpstrDeviceType="ávivideo";
    mciopen.ElementName=path;
    if ((rtrn=mciSendCommand(0,MCI_OPEN,
                             MCI_OPEN_TYPE|MCI_OPEN_ELEMENT,
                             (DWORD)(LPVOID)&mciopen))!=0L){
        mciGetErrorString(rtrn,(LPSTR)b, STRINGSIZE);
        DoMessage(hwnd,b);
        return(0L);
    }
    videoID=mciopen.wDevideID;
    mciplay.dwCallback=(DWORD)hwnd;
    if ((rtrn=mciSendCommand(videoID,MCI_PLAY,MCI_NOTIFY,
                             (DWORD)(LPVOID)&mciplay))!=0L){
        mciSendCommand(videoID, MCI_CLOSE,0,NULL);
        mciGetErrorString(rtrn, (LPSTR)b, STRINGSIZE);
        DoMessage(hwnd,b);
        return(0L);
    }
    return(1L);
}
```

En este ejemplo de la función PlayFlick, el argumento path es la trayectoria del archivo AVI que será ejecutado. El hwnd es el manejador de la ventana. Debería de existir un identificador entero global llamado videoID definido en alguna parte para guardar el valor del ID del manejador regresado por la llamada a MCI_OPEN.

Mucho del código de PlayFlick es parecido al que muestra la ejecución de WAVE. Hay dos cosas que hacer notar -el dispositivo MCI en cuestión es llamado "avivideo" y algunas de las estructuras involucradas son específicas a los archivos AVI. Las definiciones de estas estructuras estan en el archivo llamado DIGITAL.H en el directorio \WINVIDEO\ INCLUDE. No se necesita el paquete completo de Video para Windows para ejecutar los archivos AVI, pero se requiere de él para compilar alguna aplicación que lo este utilizando.

La primera llamada a mciSendCommand abre el dispositivo avivideo y regresa el valor ID para éste. El segundo llama a ejecutar el archivo AVI que se le está pasando. Como con los archivos wave y el audio de CD-ROM, la interfaz MCI notificará a la ventana que se seleccione cuando el archivo AVI que se ha empezado a ejecutar, termine por mandar un mensaje de MM_MCINOTIFY a ésta.

Habiendo ejecutado la segunda llamada a mciSendCommand, una ventana deberá aparecer en la pantalla, desplegando el video, y si el video contiene sonido, éste se desplegará si se tiene una tarjeta de sonido, y dada la característica de AVI de tener imágen y sonido intercalados.

Si se quieren modificar las características de la ventana en la cual un archivo AVI se está desplegando. Se puede hacer porporcionandole a MCI la ventana en la que se desplegará el video. Por ejemplo, si se quisiera usar el manejador para reposicionar la ventana. Aqui hay un ejemplo:

```
MCI_DGV_STATUS_PARMS mcistat;
if((rtrn=mciSendCommand(videoID,MCI_STATUS,MCI_DGV_STATUS_HWND,
                        (DWORD)(LPVOID)&mcistat)) != 0L){
    mciGetErrorString(rtrn,(LPSTR)b,STRINGSIZE);
    DoMessage(hwnd,b);
    return(0L);
}
```

Habiendo ejecutado este código, el HWND para la ventana identificada por el videoID regresado cuando se abre el dispositivo "avivideo" estará en el elemento dwreturn del objeto mcistat.

La mayoría de las llamadas para trabajar con archivos AVI son especializadas - por ejemplo MCI_DGV_STATUS_PARMS en lugar de MCI_STATUS_PARMS , que es el utilizado para audio de cd o archivos wave. Es importante tener en cuenta esto y utilizarlos cuando se esperan, ya que tienen internamente estructuras diferentes. En el manual de Video para Windows se listan las llamadas que son relevantes para trabajar con archivos AVI.

Como en los ejemplos anteriores que utilizan llamadas a MCI, se debe tener un caso en el manejador de mensajes de la ventana, que reciba la notificación de la terminación del archivo AVI que se ejecutó, para responder. En su forma más simple sería como:

```
case MM_NOTIFY:
    mciSendCommand(LOWORD(lParam),MCI_CLOSE, MCI_WAIT, NULL);
    videoID=-1;
    break;
```

La llamada deberá de cerrar el dispositivo y esperar hasta que la operación se haya completado antes de salir.

Se puede terminar la ejecución de un archivo AVI en una de dos formas: haciendo

doble click en el menu del sistema de la ventana que la contiene, o al hacer la llamada a MCI:

```
MCI_GENERIC_PARMS mcigen;
mciSendCommand(videoID,MCI_STOP, MCI_NOTIFY|MCI_WAIT,
(DWORD)(LPVOID)&mcigen);
```

El comando MCI_STOP causará un mensaje de MM_NOTIFY que será mandado a la ventana destino para el manejador especificado por videoID, justo como si hubiera terminado.

No hay buffers externos que liberar en la ejecución de un archivo AVI - todo el alamacenamiento requerido por el manejador, cuando es abierto, es liberado cuando una llamada a MCI_CLOSE se realiza.

4.3 Objetos pasivos

4.3.1 Imágenes

El manejo de imágenes en Windows, no es parte del conjunto de herramientas de desarrollo multimedia, esto es parte de Windows mismo.

Una imágen bitmap es un área rectangular que contiene bits de color de la gráfica que representan . En esta imágen cada pixel es guardado como un valor que representa el color. Y existe una paleta que representa todos los colores que la gráfica contiene.

Windows corre al mismo tiempo varias aplicaciones, cada una puede tiene su paleta de color, y para evitar que cuando haya cambios de paleta se afecte los colores del ambiente de windows, este reserva una paleta de 20 colores, los cuáles se mantienen sin cambios,

para no verse afectados por las otras aplicaciones. En un ambiente de 16 colores, estos abarcan los 16 colores reales y 4 colores generados, por alternar puntos de los 16 primeros colores. En un ambiente de 256 colores, son 20 colores genuinos de la paleta de color, con los restantes 236 colores disponibles para el uso de la aplicación.

Por la forma en que se maneja la paleta de color, si se intenta desplegar una imágen bitmap de 16 colores en un ambiente de 16 colores, que utiliza una paleta diferente a la del sistema, se deben remapear los colores de la paleta de la imagen a la paleta de windows, que puede causar corrimientos de color en algunas gráficas.

Lo mismo pasa si se intenta desplegar un bitmap de 256 colores bajo un ambiente con un driver de color de 16, aunque en este caso es más radical, y las imágenes pierden su calidad.

Algo similar sucede cuando se trata de desplegar bitmaps de 256 colores en un driver de 256 colores. Ya que el bitmap ocupa las 256 entradas de color y Windows solo tiene 236 libres, por los 20 reservados. En este caso Windows remapea 20 de los colores del gráfico para desplegarlos.

Para estos casos un método muy utilizado es el de "dither" , que consiste en usar patrones de puntos alternados para simular colores que no están disponibles de otros que si lo están.

Windows además soporta drivers de despliegue de color real, los cuales no utilizan paletas de color. Si se intenta desplegar en tales dispositivos no hay necesidad de remapear.

La profundidad de un gráfico está definidos por el dato que la representa, bajo Windows se dice que se tienen 1, 4, 8 o 24 bits de color, representando los valores de color máximo de 2, 16, 256 y 16777216 respectivamente.

En las aplicaciones que tenga la máxima calidad de su despliegue gráfico se deben tener versiones de imágenes sin "dither" para cuando se utiliza 256 colores y unas segundas imágenes con "dither" para 16 colores.

La forma de saber el número de bits soportados por el driver, es con el siguiente código:

```
HDC hDC;
int bits;
hDC=GetDC(hwnd);
bits=(GetDeviceCaps(hDC, PLANES)*GetDeviceCaps(hDC, BITSPIXEL));
ReleaseDC(hwnd, hDC);
```

Donde hwnd es el manejador de la ventana. El valor de bits será el valor de profundidad del driver.

4.3.1.1 Estructuras bitmap de Windows

Existen dos tipos fundamentales de bitmaps en Window: bitmaps dependientes de dispositivos y bitmaps independientes de dispositivo (DIB). El primer tipo son un patrón de bits guardados en memoria. Los bits por pixel son organizados en la misma forma como los que se encuentran en un dispositivos de salida en particular. Un bitmap para un tipo de dispositivo de salida no puede ser utilizado sobre otro dispositivo que utilize una diferente organización de bits por pixel.

Un bitmap independiente de dispositivo de es además un patrón de bits guardados en memoria. Sin embargo, sus bits están organizados en la forma que mejor describe la imágen codificada. Este incluye el número de colores utilizados en un bitmap, los colores que son usados y la redundancia de los datos (utilizada cuando se selecciona un formato de compresión).

El formato de un bitmap dependiente del dispositivo es conocido por el dispositivo que se esté utilizando. El formato DIB tiene una especificación que le acompaña. Esta especificación describe el alto y ancho del bitmap, el número de planos de color, el número de pixeles, el tipo de compresión usado, la resolución vertical y horizontal del dispositivo

para el cuál el bitmap fué diseñado, el número de colores actualmente utilizados y el número de colores considerados importantes para el despliegue del bitmap.

Para utilizar un bitmap independiente del dispositivo, primero debe convertirse a un bitmap dependiente del dispositivo. GDI(*Graphics Display Independent* utiliza la especificación del DIB y las capacidades del dispositivo para propiamente mapear los valores del pixel en el DIB a los propios en el bitmap dependiente del dispositivo.

La estructura interna de un bitmap varia considerablemnete, dependiendo de su profundidad de color. así también como de la tarjeta de la memoria de la pantalla. dependiendo de su profundidad de color. dimensiones, etc. Es asíque una tarjeta de despliegue es llamada "device" para Windows. Un bitmap que puede ser desplegado en un dispositivo particular es llamado un bitmap dependiente.

Un bitmap tiene además características amorfas bajo windows llamadas "device context". Cuando se debe actualizar una ventana, la función BeginPaint se le debe proporcionar el manejador del "device context" que representa la pantalla.

Un bitmap dependiente de dispositivo de 16 colores no desplegará correctamente en un dispositivo de 256 colores. De tal forma , está no es la forma en que son guardados bajo windows.

La estructura de un bitmap independiente del dispositivo es constante. este consiste de tres elementos:

- Un objeto BITMAPINFOHEADER que define el tamaño y profundidad de color del bitmap, entre otras cosas.
- Una lista de objetos RGBQUAD para definir la paleta de colores.
- Los datos del bitmap en si mismo.

Un objeto BITMAPINFOHEADER le dice a una función que quiere hacer algo con

un bitmap, cómo el bitmap es mandado. Este es el objeto en si mismo:

```
typedef struc{
    DWORD biSize;
    DWORD biWidth;
    DWORD biHeight;
    WORD biPlanes;
    WORD biBitCount;
    DWORD biCompression;
    DWORD biSizeImage;
    DWORD biXPelsPerMeter;
    DWORD biYPelsPerMeter;
    DWORD biClrUsed;
    DWORD biClrImportant;
} BITMAPINFOHEADER;
```

El campo biSize define el número de bytes utilizados por el encabezado , éste siempre es (DWORD)sizeof(BITMAPINFOHEADER). El biWidth y biHeight definen las dimensiones del bitmap en pixeles. El campo de biPlanes siempre es uno. biBitCount puede ser 1, 4 , 8 ó 24 representando el número de bits de información de color en el bitmap.

Siguiendo un objeto BITMAPINFOHEADER, se encuentran dos o más objetos RGBQUAD a menos que el archivo contenga una imágen de 24 bits, ya que estas imágenes no tienen paleta. El número RGBQUAD puede ser calculado como $2^{biBitCount}$. Cada RGBQUAD define un color en la paleta para la imágen. Un RGBQUAD se ve como :

```
typedef struct{
    BYTE rgbBlue;
    BYTE rgbGreen;
    BYTE rgbRed;
    BYTE rgbReserved;
} RGBQUAD;
```

Los campos de rgbBlue,rgbGreen y rgbRed especifican el porcentaje de luz roja, verde y azul involucrada en la definición de color. El campo de rgbReserved deberá ponerse en cero, y no es utilizada. Siguiendo los objetos RGBQUAD deben estar los datos del bitmap en sí mismo.

4.3.1.2 Despliegue de bitmaps

Hay dos funciones que nos permiten desplegar imágenes en windows. La mas simple y la más ràpida es BitBlt (Bitmap block transfer). La segunda es SetDlBitsToDevice, esta ultima no es tan rápida como BitBlt, pero utiliza memoria menos vorazmente, y es capaz de desplegar bitmnps grandes.

Un ejemplo simple de BitBlt es la siguiente:

```
HBITMAP hOldBitmap;
HDC hMemoryDC;
BITMAP bitmap;
if((hMemoryDC=CreateCompatibleDC(hdc))!=NULL){
    hOldBitmap=SelectObject(hMemoryDC,hBitmap);
    if (hOldBitmap){
        GetObject(hBitmap, sizeof(BITMAP),
        (LPSTR)&bitmap);
        BitBlt(hdc, 0, 0, bitmap.bmWidth,
        bitmap.bmWeight,
        hMemoryDC, 0, 0, SRCCOPY);
        SelectObject(hMemoryDC, hOldBitmap);
    }
    DeleteDC(hMemoryDC);
    }
```

En este ejemplo hBitmap es un manejador de un bitmap -la combinación de estructuras

tratadas antes sin un BITMAPFILEHEADER al principio. El objeto hdc es un manejador del dispositivo de contexto proporcionado por el caso de WM_PAINT del manejador de mensajes de la ventana en la cuál el bitmap será desplegado.

La función CreateCompatibleDC genera un contexto de dispositivo para un bloque de memoria que está estructurado de la misma forma como el manejador HDC del manejador de la pantalla pasada a éste. En esta aplicación el contexto del dispositivo es utilizado como el espacio de trabajo en el cuál se ensamblará la versión dependiente del dispositivo del bitmap independiente del dispositivo, para ser desplegado.

La función SelectObject selecciona el bitmap para ser desplegado dentro del nuevo contexto del dispositivo, esto es, causa una copia del dispositivo dependiente al bitmap que es creado.

La llamada a GetObject es una forma simple de obtener las dimensiones de un bitmap referenciado por el manejador. Temporalmente asegura el bitmap, copia al principio de éste un objeto bitmap, y entonces lo libera.

Finalmente, BitBlt despliega el bitmap. Esta función copia bloques rectangulares de bits de un dispositivo a otro, aquí el por qué el bitmap original fue convertido a un contexto de dispositivo basado en memoria. En esta aplicación, el primer argumento de BitBlt, es el contexto del dispositivo en el cual el bitmap será desplegado. El quinto y sexto argumentos son la localización en pixeles de la esquina superior izquierda del bitmap relativo a la esquina superior izquierda de la ventana. En este caso, el bitmap deberá ser desplegado empezando en el punto (0,0) de la ventana.

El último argumento es una constante para especificar cómo los pixeles fuentes son relacionados a los pixeles que actualmente existen en el bitmap destino. La constante SRCCOPY le dice a la función que copie los pixeles fuente sobre los pixeles destino. Otras opciones son :

SRCAND Realiza operación de AND entre bitmap fuente y destino.

SRCERASE OR-Exclusivo de los bitmaps fuente y destino.

SRCPAINT OR de los bitmaps fuente y destino.

Si la aplicación despliega imágenes pequeñas, la función BitBlt es una buena forma de manejarlas.

Cunado se utilizan imágenes muy grandes se debe ocupar la función SetDlBits que copia un bitmap a un contexto de dispositivo. Es considerablemente más flexible que BitBlt, por ejemplo deja copiar secciones de un bitmap, aunque es un poco más complicada para usar.

En el siguiente ejemplo, hdc es un manejador del contexto del dispositivo que será actualizado y picture es un manejador de un bitmap. Width y depth representan las dimensiones del bitmap. El valor bits es el número de bits de color que soporta.

```
LPBITMAPINFO bmp;
RGBQUAD far *palette;
HANDLE hPal=NULL;
LPSTR image;
LOGPALETTE *pLogPal;
int i,j,n;
if ((bmp=(LPBITMAPINFO)GlobalLock(picture))!=NULL){
    n=<< bits;
    j=min(n,256);
    palette=(RGBQUAD far *)((LPSTR)bmp+
        (unsigned int)bmp → bmiHeader.biSize);
    image=(LPSTR)bmp+(unsigned int)bmp → bmiHeader.biSize+
        (j*sizeof(RGBQUAD));
    if((pLogPal=(LOGPALETTE *)malloc(sizeof(LOGPALETTE)+
        (j*sizeof(PALETTEENTRY)))) != NULL){
```

```
            pLogPal → palVersion=0x0300;
            pLogPal → palNumEntries=j;
            for (i=1; i<j; i++ ) {
                pLogPal → palPalEntry[i].peRed=palette[i].rgbRed;
                pLogPal → palPalEntry[i].peGreen=palette[i].rgbGreen;
                pLogPal → palPalEntry[i].peBlue=palette[i].rgbBlue;
                pLogPal → palPalEntry[i].peFlags=0;
            }
            hPal=CreatePalette(pLogPal);
            free(pLogPal);
            SelectPalette(hdc,hPal,0);
            RealizePalette(hdc);
        }
        SetDIBitsToDevice(hdc,0,0,width,depth, 0, 0, 0, depth,
        image, bmp, DIB_RGB_COLORS);
        if (hPal !=NULL)DeleteObject(hPal);
            GlobalUnlock(picture);
    }
```

La llamada a SetDIBitsToDevice copia la información del bitmap a la ventana en la cual será desplegada. El primer argumento a SetDIBitsToDevice es un manejador al contexto del dispositivo para recibir el bitmap. El segundo y tercer argumentos especifican las coordenadas de la ventana en la cual el bitmap aparecerá - donde la esquina superior izquierda de la imágen deberá proporcionarse. El cuarto y quinto argumentos especifican las dimensiones de el bitmap fuente en pixeles. El sexto y séptimo argumentos especifican la esquina superior izquierda de el rectángulo que define el área del bitmap fuente que será copiado a la ventana destino. El octavo argumento es el número del primer scan line en el bitmap.

El noveno argumento es un apuntador a los datos del bitmap para ser copiados. El décimo argumento es un apuntador a la estructura BITMAPINFO, la cuál es, en efecto, un BITMAPINFOHEADER seguido por arreglos de RGBQUAD.

Capítulo 5

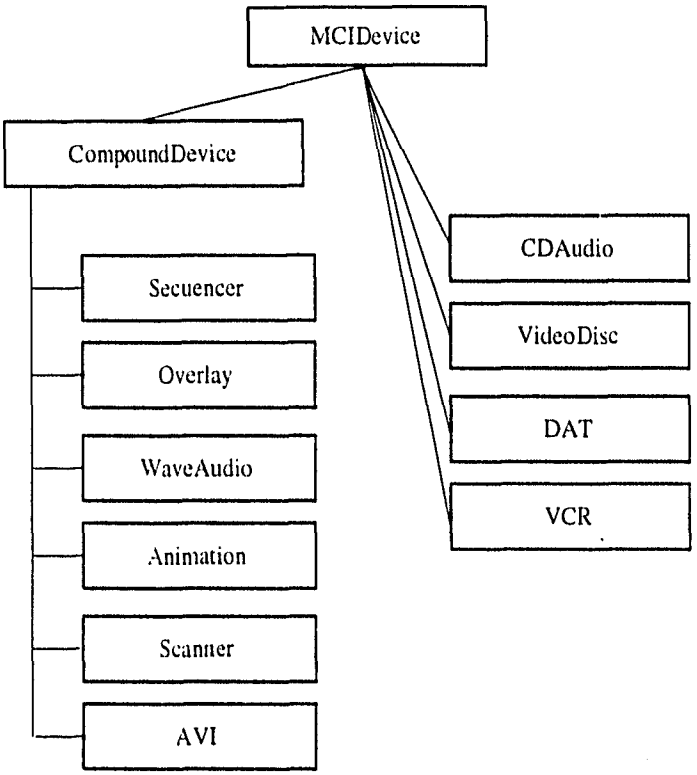
Biblioteca de clases MCI

Los comandos MCI son potentes y flexibles pero tienen la desventaja de que su propósito es tan general que resultan engorrosos para manejar un dispositivo en particular. Es decir, existen muchas estructuras, campos y mensajes; pero sólo una parte se utiliza para trabajar un dispositivo en particular. De aquí nació la idea de crear una biblioteca de objetos que encapsule las características de cada uno de los medios, basadas en las funciones que nos proporciona MCI.

5.1 Diseño

Como se vió en el capítulo anterior, todos los comandos son clasificados dentro alguna de cuatro categorías: sistema, requerido, básico y extendido. Los comandos específicos para los primeros dos tipos (tales como open, close y status) son aceptados por todos los tipos de dispositivos y por lo tanto son buenos candidatos para ser definiciones de las funciones miembro en la clase base de la jerarquía de MCI. Los comandos básicos incluyendo el *play*, *stop* y *seek*, que se pueden utilizar por la mayoría de los dispositivos se pueden colocar cerca de lo alto del árbol. Los comandos extendidos que soportan dispositivos específicos

son implantados más abajo en la jerarquía. La figura siguiente muestra la jerarquía de objetos MCI.



Jerarquía de clases MCI

El nodo raíz es la clase MCIDevice, que sirve como la clase base para las clases CompoundDevice y todas las clases de dispositivos simples(CDAudio, Animation, etc). En esta clase se encuentran los comandos básicos tales como open/close/play, los comandos de estado para preguntar sobre el estado del dispositivo, conjunto de comandos para configuración, y otras funciones misceláneas.Todas con la característica de ser aceptadas por

la mayoría de los dispositivos. Los datos que se necesitan para definir a la clase son (todos privados) :

- Un entero identificador del dispositivo (id)
- Un valor de status de error
- Un apuntador a una cadena al nombre del dispositivo
- Un manejador opcional a la ventana padre

La declaración de la clase puede verse en el archivo biblio.h que se encuentra en el *Apéndice B.*

El constructor, no abre automáticamente el dispositivo,si lo hicieramos de esta forma, al abrirse sería inaccesible a otra aplicación, ya que lo libera hasta que el objeto es destruido. Esto también se cumple para todas las clases derivadas. Un dispositivo no es abierto hasta que explícitamente se le indica con una llamada a Open( ). Este es cerrado también a través de una llamada a la función miembro Close( ) o cuando el objeto es destruido y el destructor es invocado. Si construimos un objeto CompoundDevice con un nombre de archivo, es tratado como si éste hiciera una llamada a Open( ) y el archivo es inmediatamente abierto. Esta forma de manejar el abrir el dispositivo es importante para no mantener elementos inaccesibles a otras aplicaciones, obteniendo el mejor funcionamiento del ambiente multitarea que se esté trabajando.

El constructor para MCIDevice, y cualquier dispositivo simple derivado (tal como CDAudio), toma un solo argumento, una cadena especificando el nombre apropiado del dispositivo MCI. Se proporciona una cadena de entrada para cada uno de los tipos de dispositivos simples, de tal forma raramente se tenga que especificar este valor. La única excepción podría ser sobre un sistema utilizando nombres de dispositivos adicionales o no estándar, que no concuerdan con las cadenas usuales de nombres de dispositivos MCI definidos en la seccion [mci] del archivo SYSTEM.INI. MCIDevice no es una clase abstracta y puede así construirse y usarse directamente. Ésta es diseñada, sin embargo, para servir

como la base para clases derivadas, soportando tipos de dispositivos únicos.

La mayoría de las funciones miembro tienen un mapeo directo al correspondiente comando MCI. Y la forma en que se implementan hace que requieran pocos argumentos a diferencia de si usaramos las interfaces de comando-mensaje o comando-cadena, debido a que alguna parte de la información necesaria se encapsula en el objeto mismo. Cada función miembro, inicializa alguna estructura de datos, pone las banderas apropiadas y los argumentos, y hace una llamada al mciSendCommand().

El código de retorno de la función se regresa al cliente del objeto y además, se almacena en orden para mantener un estado de error, por si se requiere una referencia posterior. La función miembro ErrorMessage() llama a mciGetErrorString() para obtener la descripción del error y desplegar una caja de diálogo con el mensaje de error, para el usuario.

Las clases que se definen muestran uno de los beneficios de utilizar C++: que son los destructores para hacer la limpieza automática. Es importante que las aplicaciones cierren apropiamente todos los dispositivos MCI y los archivos que han abierto. Al utilizar el mecanismo de destructores de C++, se puede provocar el cierre de cualquier dispositivo abierto antes de que una aplicación termine, y así proporcionar confiabilidad y robustez.

MCIDevice proporciona las funciones virtuales Set(), Status() y GetDevCaps() que son operaciones de acceso, que tienen la característica de estar definidas como protegidas, o sea no pueden ser llamadas directamente. Si fueran públicas forzaría al usuario, a estar familiarizado con las numerosas constantes asociadas a los mensajes MCI_SET y MCI_STATUS, necesarios para cada opción específica. En su lugar, las operaciones de acceso son usadas como el mecanismo dentro de las funciones en línea definidas públicamente para cada una de las opciones del conjunto de *set* y *status*. Por ejemplo, la funcion *Length*, la cual retorna el tamaño del dispositivo actual, usa la función Status() con la opcion MCI_STATUS_LENGHT para obtener su valor.

Otra función que se implementa es SetParent(), que es usado para asignar el manejador de ventana designado como el padre de éste objeto MCI. La función miembro Play() revisa si una ventana padre ha sido asignada a él y si es así, guarda este manejador, en el bloque

de parámetros dado para mciSendCommand( ) y pone valor correspondiente de la bandera MCI_NOTIFY. Este causa que MCI mande un mensaje de MM_MCINOTIFY a la ventana especifica, cuando la operación es finalizada.

Se ha tratado que el diseño sea flexible, extendible y sobre todo práctico. No todas las funciones de los dispositivos han sido implementadas, sin embargo ampliarlas para dispositivos adicionales es un proceso fácil.

5.2 Dispositivos simples MCI

Todas las funciones básicas para los dispositivos simples son proporcionadas en la clase base MCIDevice. Así CDAudio se puede definir como una clase simple por que no necesita proporcionar funciones adicionales que las de la clase MCIDevice. Este dispositivo puede, en su lugar, depender de las operaciones heredadas de su padre. El conjunto de comandos MCI para CDAudio define unas pocas opciones adicionales para algunos de los comandos básicos. Los cuáles son Eject( ), para sacar el disco. SetTimeFormatMSF( ) y SetTimeFormatTMSF( ) para poner el formato del tiempo (T=Tracks, M=minutes,S=second, F=Frames). Cada uno es implementado como una función en línea, que pasa las banderas apropiadas para la función Set( ). Al permitir poder especificar el formato del tiempo, podría inadvertidamente cambiar algún dispositivo a un formato erróneo, así un vistazo a la definición de la clase, nos permite ver los formatos válidos para cada dispositivo.

Este dispositivo de CDAudio puede ser abierto, cerrado, parado o interrogado para su status. Todas las operaciones comúnes son heredadas de la clase base MCIDevice. Otros dispositivos simples tales como VideoDisc y VCR pueden ser implantados similarmente al derivar de MCIDevice, especificando las cadenas de tipo de dispositivo para el constructor y creando funciones específicas que sean requeridas para cada dispositivo.

5.3 Dispositivos Compuestos MCI

Un dispositivo compuesto es un dispositivo MCI que utiliza archivos. Por lo tanto la clase CompoundDevice y sus descendientes deben ser capaces para manejar un nombre de archivo sobre el objeto abierto. Al nombre de archivo para un dispositivo compuesto es conocido como elemento de dispositivo. El constructor para CompoundDevice toma dos argumentos adicionales : el nombre del archivo y el nombre del tipo de dispositivo. El segundo argumento es inmediatamente pasado al constructor de la clase base . El primer argumento, si es dado, es guardado internamente y es usado para abrir el archivo. Esta propuesta permite el modelo de iostream, en el cual un nombre de archivo pasado al constructor automáticamente abre el archivo. La implementación podría fácilmente ser cambiada para no abrir el archivo y requerir una subsecuente llamada a Open( ). No requerir un nombre de archivo permite al objeto ser contruido primero y más tarde proporcionar el nombre del archivo con la llamada a Open( ).

Las funciones virtuales de Open( ) y Close( ) son redefinidas para tratar con los nombres de archivos. Open( ) salva el nombre del archivo como parte de los datos del objeto y entonces pasa este en el campo lpstrElementName de la estructura mciOpenParms dada a mciSendCommand( ). Close( ) utiliza MCIDevice::Close para cerrar el dispositivo y entonces liberar la memoria asociada con el nombre del archivo.

Un ejemplo de la derivación que se lleva a acabo desde el CompoundDevice es la clase Wave, que como vimos anteriormente, es audio digital. Esta clase no necesita sobreescribir la mayoría de las funciones virtuales heredadas de CompoundDevice. Las funciones tales como Play, Stop, Resume y Seek, pueden ser utilizadas tal cual. Se han agregado funciones adicionales de estatus para averiguar sobre el formato del archivo actual: Channels( ), el cual regresa 1 si el archivo es mono o 2 si es estereo. BitPerSample( ), el cual regresa 8 o 16; y SamplesPerSecond( ) que regresa 11025, 22050 0 44100 que informa de la tasa de muestreo.

Otro dispositivo compuesto es el implementado para manejar archivos AVI donde se

implementan gran número de funciones, ya que el driver que proporciona Windows para AVI, proporciona extensiones de comandos al dispositivo de Video. La función Window( ), fue adicionada, para permitir la reproducción del video en una ventana específica, más que en la ventana de que es creada en el ambiente. PutDestination( ), que permite especificar un área rectangular dentro de la ventana padre, para la posición del video. Step( ) mueve hacia adelante o hacia atrás, un cierto número de cuadros; Seek( ) mueve a una posición determinada; Update( ), que despliega el cuadro actual; y Signal( ), el cuá informa al padre cuando una posición ha sido encontrada.

5.4 Compilación

Cuando se utilizan bibliotecas, el ligador las localiza y las adhiere al archivo ejecutable. Si la biblioteca es muy grande, obviamente el tamaño del programa crece, y si hay varias aplicaciones que ocupan la misma biblioteca cada uno tiene el mismo código repetido. Una solución a esto, es utilizar DLL (Dynamic Link Libraries). Que son ligadas al programa en el momento de ejecución. Esto significa que mantiene sólo una copia en memoria y utiliza esta , para todos los programas que la necesiten.

Para crear la biblioteca como DLL, se crea un archivo biblio.h (Véase Apéndice B) que contiene la definición de las clases, este archivo debe ser incluido en el fuente del DLL que estamos creando, biblio.cpp. Se le agrega a la definición de las clases la palabra _EXPORT . Esto le dice al compilador que las direcciones de las clases son exportadas, y por tanto se hacen disponibles a las aplicaciones que utilicen la biblioteca.

Además del código fuente, que se encuentra en biblio.cpp (véase Apéndice B) Windows requiere que se le proporcione un punto de entrada y salida de las rutinas en el DLL. Las funciones que realizan esto se encuentran en el archivo bibliosh.cpp, y son: LibMain y WEP. La primera es llamada por Windows cuando la biblioteca es cargada en memoria y WEP cuando la desecha. Y son regularmente utilizadas la primera para inicializar vnriables globales de la biblioteca y la segunda para liberar memoria de elementos que se hayan pedido.

El archivo biblio.def, que es utilizado para definiciones del compilado y ligado, debe llevar como primer enunciado la palabra LIBRARY, así se indica que es un DLL en lugar de un archivo ejecutable normal. Como es obvio, hay que indicar dentro del ambiente de compilación, que el código generado será un DLL, seleccionando Application|Options|Windows DLL, en el menú.

Para utilizar dicha biblioteca, así generada dentro de una aplicación, tan sólo se requiere incluir el archivo biblio.h, además en Options|Compiler|CodeGeneration|Defines hay que poner la constante _CLASSDLL, indicando con esto que las clases que se utilizan en el programa se encuentran dentro de un DLL. Si se está utilizando owl, las clases de éste también deben manejarse con su opción de DLL.

Por último dentro del proyecto, se debe incluir el archivo biblio.lib, que se generó a partir de biblio.dll, utilizando la utilería IMPLIB.EXE. Este archivo biblio.lib, resuelve las referencias a las llamadas a rutinas dentro de la biblioteca que serán llamadas a la hora de ejecución.

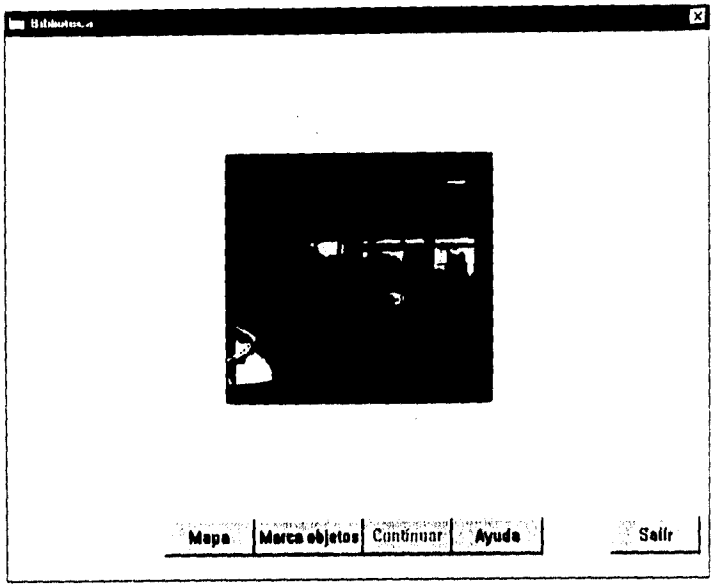
Para correr la apliración se debe tener en la trayectoria de búsqueda de Windows el lugar donde se encuentra biblio.dll.

5.5 Ejemplo de aplicación

Para mostrar el manejo de los dispositivos utilizando la biblioteca de objetos, se creó una aplicación donde se muestran elementos que no se podrían manejar con programas de autoría, como son, video interactivo por selección de regiones.

El programa consiste de una navegación a través del video de la Biblioteca de la

Facultad de Ciencias.



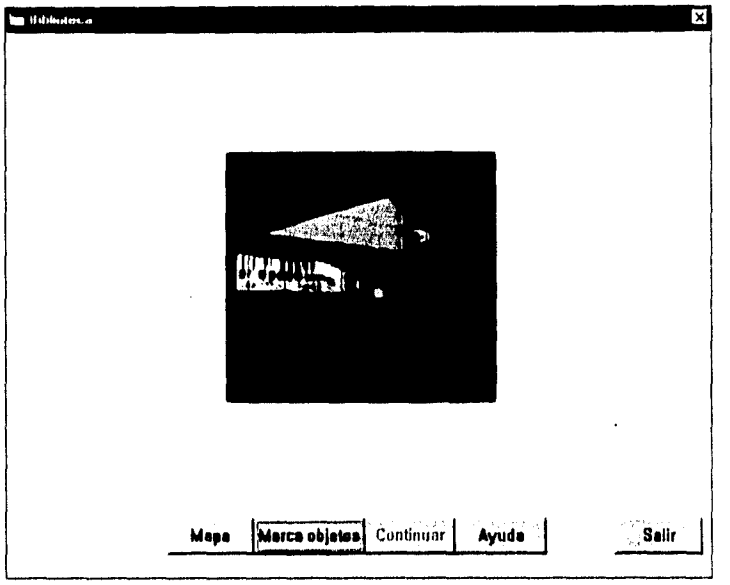
Navegación por la biblioteca

Recorriendo la escalera que une los diferentes niveles de la biblioteca. A partir de este camino, y en diferentes secciones del video, el visitante puede escoger objetos que son parte del mismo video y que lo llevan a otros lugares, por ejemplo, si selecciona la puerta de la biblioteca, cambia el camino hacia la biblioteca, la muestra y sigue el recorrido.

En cualquier momento se puede cambiar de dirección, es decir, si se está subiendo una escalera, al oprimir el botón derecho del ratón, se corre el video que recorre la misma escalera, pero en descenso.

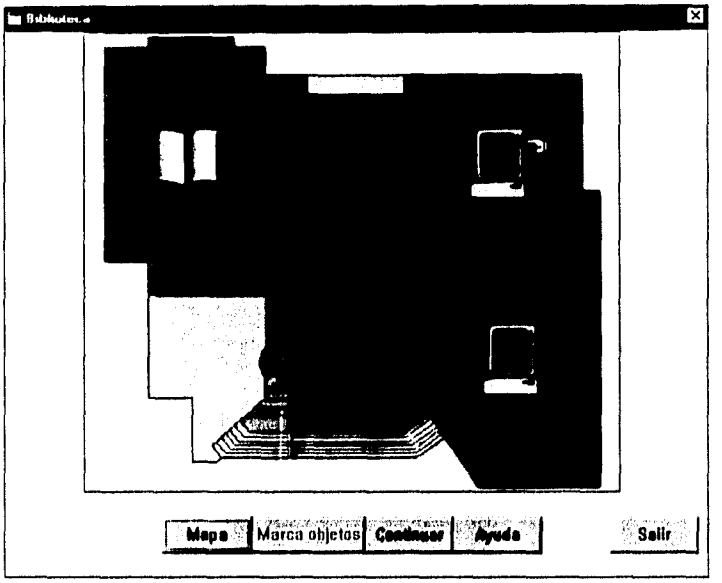
En las secciones de video que contienen objetos para seleccionar, aparece un cursor en forma de mano. Los objetos que llevan a alguna otra parte dentro del video, aparecen con

un fondo negro, si se oprime el botón de **Marca objetos**. En estos videos no funciona la opción de cambio de dirección.



Objeto marcado para seleccionar

El visitante además puede ir viendo la posición en la que se encuentra dentro de la biblioteca, basta con oprimir el botón de **mapa**, entonces se muestra un bitmap esquemático de la biblioteca, y con una imágen de pies, muestra el lugar en donde se ubica el usuario. Si se oprime el botón derecho sobre algún lugar dentro del mapa, también cambia la dirección dentro del recorrido.



Mapa de posición del usuario

5.5.1 Implementación

Como primer paso se grabó el video de la biblioteca con sus diferentes caminos alternativos. La grabación se hizo con una cámara de video casera, y la digitalización a formato AVI, se realizó utilizando la tarjeta Captivator. La velocidad de despliegue es de 15 frames por segundo y se tiene un formato de 240 x 204 pixeles, con 24 bits de color. Esto implica que el programa sólo se verá bien en una máquina que maneje 16 millones de colores. El video original fue seccionado en las diferentes partes del paseo, para que su manejo fuera más rápido, ya que tener un video demasiado grande en la máquina, provoca que sea muy lento el recorrido.

La implementación del camino se hizo con una lista doblemente ligada, donde cada nodo es un objeto de la clase:

```
class NODO {
public:
    AVI *apvideo;       // Apuntador al video
    int mapa;       // Indice dentro de los bitmaps
    int tipo;       // Tipo: MUESTRA, CAMINA, SELECCIONA
    char *nombre;       // Nombre del video en disco
    int Sel;        // Indice dentro de los frames
    NODO( AVI *video, int mapa, int tipo, char *cad, int num);
    ~NODO();
};
```

Este nodo contiene un apuntador a un objeto AVI (apvideo), que es el video asociado a ese lugar. Durante el programa se mantiene una variable llamada Lugar, que apunta al nodo que contiene la información de la posición actual. al inicio del programa esta variable se inicializa apuntado al primer nodo de la lista, y mandando un mensaje de WM_CORREVIDEO, a la ventana principal, la cuál ejecuta la función Corre, que se encarga de definir la ventana, la posición dentro de la ventana y activar el mensaje de notificación. Si el usuario no interactún con el video, es decir si no hay cambio de dirección, al terminar de ejecutarse un video, avisa al programa que ya terminó de ejecutarse a través del mensaje WM_MCINOTIFY, dispara el siguiente video en la lista y, dado que es circular, esto sigue hasta que el usuario decide salir del programa.

Dentro de esta clase, se encuentra un elemento tipo, que puede tener los valores de MUESTRA, SELECCIONA y CAMINA, y sirve para conocer que funciones se llevan a cabo cuando hay interacción del usuario. Por ejemplo, cuando el video es de tipo MUESTRA y se oprime el botón izquierdo, sólo adelanta el video; pero si fuera de tipo CAMINA, cambiaría de dirección.

El elemento mapa es un índice a un arreglo, cuyos elementos son una lista de índices que localizan dentro de un arreglo que contiene todos los bitmaps, que se necesitan en el

programa, las imágenes que sobrepuestas forman el lugar donde se localiza el usuario. Si se manejarán un bitmap por cada localización del usuario, dado que los bitmaps están realizados en 24 bits, el tamaño de cada uno sería de 547254 bytes al compilarse el programa. El ejecutable incrementa su tamaño demasiado, además al trabajar con ellos ocupan mucha memoria, y el despliegue se hace lento considerablemente. Para evitar esto se generarón los diferentes bitmaps que correspondieran a los lugares en que el usuario podría estar, remarcando los sitios de interés como es la sala de consulta de libros -representada con un libro -, las salas de cómputo -imágen de computadora-, sala de lectura -imágen de mesita con lápiz y libro-, y la posicion del visitante- imágen de pies-. De estas imágenes se recortaron, las partes la computadora, el libro , la mesita y las diferentes posiciones del pie, generando varios bitmaps pequeños. Estos bitmpas se cargan al principio del programa, y se tiene un arreglo que los apunta. De esta forma para mostrar la posición del usuario, se recorre la lista de índices, correspondiente al lugar actual, y sobre una imágen que sólo contiene un esquema de la biblioteca, se van sobreponiendo, los bitmaps que indica la lista. De eta forma se utiliza menos memoria y el acceso a los bitmaps es más rápido.

El elemento Sel es el indice al arreglo que contiene listas de NODOFRAMES :

```
class NODOFRAMES {
public:
    int NumFrame;
    TLista<int> *Poli;
    NODOFRAMES(int num);
    ~NODOFRAMES();
};
```

Donde cada nodo tiene en NumFrame, el cuadro en que se puede seleccionar, en el elemento Poli, una lista de indices a un arreglo de poligonos, que describen al objeto dentro del video, que se puede seleccionar. Esto es, porque por cada cuadro se pueden tener varios polígonos, que cada uno lleve a diferentes lugares, así los polígonos se definen por la clase:

```
class NODPOL{
public:
```

```
    POINT *poligono;
    int quelugar;
    int numero;
    NODPOL();
    NODPOL(int num,int x1,int y1,int x2,int y2,int x3,int y3,
        int x4,int y4,int lugar);
    NODPOL(int num, int x1,int y1,int x2,int y2,int x3,int y3,
        int x4,int y4,int x5,int y5, int lugar);
    NODPOL(int num,int x1,int y1,int x2,int y2,int x3,int y3,
        int x4,int y4,int x5,int y5,int x6,int y6,int lugar);
    ~NODPOL();
};
```

Para cada polígono se guarda en número cuantos vértices son, en poligono se almacenan los pares coordenados, y quelugar es un indice de un arreglo a apuntadores a nodos de la lista camino. De esta forma cuando el usuario teclea sobre el video, se hace una pausa, se obtiene el numero de cuadro, y se busca en la lista asociada a ese video, si se encuentra el cuadro, entonces con la posicion del ratón, se busca si esta dentro de la región, definida por los polígonos asociados. Se actualiza la variable Lugar a la nueva posición que le corresponde según lo indique la variable quelugar del polígono al que pertenece.

Conclusiones

Con la creación de la biblioteca se ha logrado manejar los dispositivos de una forma más sencilla y robusta, ya que el programador no tiene, que manejar la gran cantidad de mensajes y estructuras, necesarias para cada dispositivo, además de la ventaja que ofrece el trabajar objetos, limpiando memoria cuando los destructores son invocados.

Un resultado importante es que gracias al manejo dinámico de la biblioteca, es decir su implementución como DLL, el código del archivo ejecutable se reduce considerablemente, ya que no pega todo el código y además sólo una copia del éste se mantiene en memoria, haciéndolo disponible para otras aplicaciones, logrando con esto, optimización de memoria, una parte importante para cualquier aplicación.

Otro factor importante es la adición de nuevos dispositivos como una tarea fácil a partir del código de la biblioteca ya existente, de tal forma que cualquier otro dispositivo nuevo puede añadirse de una forma inmediata, tan sólo con heredar de las clases bases, el tipo de dispositivo. Gracias a la expansibilidad de la biblioteca.

Finalmente se consideró importante el haber mostrado la gran interactividad que se logra a través del desarrollo del paseo interactivo a través del video, por ser una de las características más difíciles de conseguir en una aplicación multimedia, y que se logró a través del uso de definición de regiones, y del manejo del video a través de MCI, utilizando la biblioteca creada.

Apéndice A

Sumario de comandos

Las siguientes tablas presentan un sumario de los comandos utilizados en la interface comando-mensaje.

Comandos del sistema

MCI procesa directamente estos comandos más que pasarlos a MCI

Comando	Descripción
MCI_BREAK	Define una tecla de break para un dispositivo MCI
MCI_SYSINFO	Regresa información acerca de los dispositivos MCI

Comandos Requeridos

Deben ser soportados por todos los dispositivos.

Comandos	Descripción
MCI_CLOSE	Cierra el dispositivo
MCI_GETDEVCAPS	Obtiene las capacidades del dispositivo
MCI_INFO	Obtiene infortmación textual del dispositivo
MCI_OPEN	Inicializa el dispositivo
MCI_STATUS	Obtiene información de estado desde el dispositivo

Comandos básicos

El uso de estos comandos es opcional para los dispositivos.

Comandos	Descripción
MCI_LOAD	Carga los datos desde un archivo en disco
MCI_PAUSE	Detiene la ejecución
MCI_PLAY	Empieza la transmisión de datos
MCI_RECORD	Empieza la grabación de datos
MCI_RESUME	Reactiva la ejecución o grabación de un dispositivo en pausa
MCI_SAVE	Salva los datos a a archivo en disco
MCI_SEEK	Posiciona hacia adelante o hacia atrás
MCI_SET	Actualiza el estado de operación del dispositivo
MCI_STATUS	Obtiene información de estado acerca del dispositivo
MCI_STOP	Para la ejecución o grabación

Comandos extendidos

Estos comandos son especificos para ciertos dispositivos

Comando	Dispositivo	Descripcion
MCI_CUE	waveaudio	Prepara para ejecución o grabación
MCI_DELETE	waveaudio	Borra un segmento de datos desde el elemento del medio
MCI_FREEZE	overlay	Desabilita adquisición de video al *frame buffer*
MCI_PUT	animation	Define las ventanas de fuente y destino overlay
MCI_REALIZE	animation	Le dice al dispositivo que seleccione y trabaje con su paleta dentro del contexto de la ventana desplegada
MCI_STEP	animation videodisc	Brinca la ejecución de uno o más fr ames hacia adelante o hacia atrás.
MCI_UNFREEZE	overlay	Permite al *frame buffer* adquirir datos de video
MCI_UPDATE	animation	Repinta el frame actual dentro del contexto de despliegue
MCI_WHERE	animation overlay	Obtiene el rectangulo que define las áreas de fuente, destino o cuadro.
MCI_WINDOW	animation overlay	Controla las opciones de la ventana de despliegue tales como el titulo

Para una descripción completa de cada uno de estos comandos, ver el directorio de mensajes en *Multimedia Programmer's Reference*[4].

Apéndice B

Código de la biblioteca de objetos MCI

biblio.h

```
#include < windows.h>
#include <MMSYSTEM.h>
#include <stdio.h>
#include <owldefs.h>
#include "d:\winvideo\include\digitalv.h"
#include "d:\winvideo\include\mciavi.h"

#define MCI_ERR_NOT_OPEN          0x1
#define MCI_WARN_ALREADY_OPEN     0x2 // Mensajes de error
#define MCI_ERR_NOT_DEVICE        0x3 // adicionales
#define MCI_ERR_NOT_DEVICENAME    0x4 // para la biblioteca
#define MCI_ERR_NOT_PARENT        0x5
```

```
const current = - 00002.0L;
const start   = - 00001.0L;
const end     = - 00003.0L;


class _EXPORT MCIDevice {

private:
        LPSTR   lpszDeviceType; //Tipo de dispositivo
        UINT    wDeviceID;      // Identificador del dispositivo
        HWND    hWndParent;     // La ventana que maneja sus mensajes
        DWORD   dwErrState;     // Mensaje de error

public:
        MCIDevice(LPSTR lpszDevice);
        ~MCIDevice();
        LPSTR Info(DWORD dwFlags);
        virtual DWORD Open(LPSTR lpszDevice);
        virtual DWORD Close();
        DWORD Stop();
        DWORD Pause();
        DWORD Resume();
        DWORD Play(LONG lFrom, LONG lTo);
        DWORD Seek(LONG lTo);
        DWORD StopAll();
        DWORD Length(){
                return Status(MCI_STATUS_ITEM,
                        MCI_STATUS_LENGTH,0L);};
        DWORD DeviceType(){
                return GetDevCaps(MCI_GETDEVCAPS_DEVICE_TYPE);};
        DWORD Modo(){
                return Status(MCI_STATUS_ITEM,
                        MCI_STATUS_MODE, 0L);};
```

```
        DWORD Position(){
                return Status(MCI_STATUS_ITEM,
                        MCI_STATUS_POSITION,0L);};
        void ErrorMessageBox();
        BOOL IsCompoundDevice(){
                return GetDevCaps(MCI_GETDEVCAPS_COMPOUND_DEVICE);};
        DWORD SetParent(HWND hWnd){ return hWndParent=hWnd;};
        friend class _EXPORT CompoundDevice;
        friend class _EXPORT Secuencer;
        friend class _EXPORT Overlay;
        friend class _EXPORT Wave;
        friend class _EXPORT Animation;
        friend class _EXPORT Scanner;
        friend class _EXPORT AVI;
        friend class _EXPORT CdAudio;
        friend class _EXPORT Videodisc;
        friend class _EXPORT DAT;
        friend class _EXPORT VCR;

protected:
        virtual DWORD Set(DWORD dwFlags, DWORD dwExtra);
        virtual DWORD Status(DWORD dwFlags,
                        DWORD dwItem, DWORD dwExtra);
        virtual DWORD GetDevCaps(DWORD dwItem);
        void SetDeviceType(LPSTR lpszDevice);
};
```

```
class _EXPORT CompoundDevice: public MCIDevice{
        LPSTR lpszFileName;

public:
        CompoundDevice(LPSTR lpszFile, LPSTR lpszDevice);// Constructor
         CompoundDevice(); // Destructor
        DWORD Open(LPSTR lpszFile);
        DWORD Close();
        friend class _EXPORT AVI;
        friend class _EXPORT Wave;
        friend class _EXPORT Animation;
        friend class _EXPORT Secuencer;
        friend class _EXPORT Overlay;
        friend class _EXPORT Scanner;
};



/*************************** Dispositivos simples ********************/



class _EXPORT CDAudio: public MCIDevice{
public:

        CDAudio(LPSTR lpszDevice):MCIDevice(lpszDevice){
                        if (!lpszDevice) SetDeviceType("CDAudio");};
        ~CDAudio(){};
        void SetTimeFormatMSF(){
                        Set(MCI_SET_TIME_FORMAT,MCI_FORMAT_MSF);};
        void SetTimeFormatTMSF(){
                        Set(MCI_SET_TIME_FORMAT,MCI_FORMAT_TMSF);};
};
```

```
class _EXPORT VideoDisc: public MCIDevice{
public:
        VideoDisc(LPSTR lpszDevice):MCIDevice(lpszDevice){
                        if (!lpszDevice) SetDeviceType("VideoDisc");};
        ~VideoDisc(){};
};


class _EXPORT DAT: public MCIDevice{
public:
        DAT(LPSTR lpszDevice):MCIDevice(lpszDevice){
                        if (!lpszDevice) SetDeviceType("DAT");};
        ~DAT(){};
};


class _EXPORT VCR: public MCIDevice{
public:
        VCR(LPSTR lpszDevice):MCIDevice(lpszDevice){
                        if (!lpszDevice) SetDeviceType("VCR");};
        ~VCR(){};
};
```

```
/************************ Dispositivos compuestos ******************/


class _EXPORT AVI:public CompoundDevice{
public:

        AVI (LPSTR lpszFile, LPSTR lpszDevice):
                CompoundDevice(lpszFile,lpszDevice){};
        DWORD Update(HDC hdc);
        DWORD PutDestination(RECT &rect);
        DWORD Signal(DWORD dwPosition);
        DWORD Configure( );
        DWORD Cue(DWORD dwTo);
        DWORD Step(DWORD dwFrames, BOOL bReverse);
        DWORD SetAudioVolume(DWORD dwVolume);
        DWORD SetSpeed(DWORD Speed);
        DWORD Window(HWND hWnd);
        DWORD Position( );
        ~AVI( ){};
};

class _EXPORT Wave: public CompoundDevice
{
public:
        Wave(LPSTRlpszFile, LPSTR lpszDevice):
                CompoundDevice(lpszFile,lpszDevice){};
        DWORD Channels( );
        DWORD BitsPerSample( );
        DWORD SamplesPerSecond( );
        ~Wave( ){};
};
```

```
class _EXPORT Animation: public CompoundDevice
{
public:
        Animation(LPSTR lpszFile, LPSTR lpszDevice):
                        CompoundDevice(lpszFile,lpszDevice){};
        DWORD Window(HWND hWnd);
        ~Animation(){};
};

class _EXPORT Secuencer: public CompoundDevice
{
public:
        Secuencer(LPSTR lpszFile, LPSTR lpszDevice):
                        CompoundDevice(lpszFile,lpszDevice){};
        ~Secuencer(){};
};

class _EXPORT Overlay: public CompoundDevice
{
public:
        Overlay(LPSTR lpszFile, LPSTR lpszDevice):
                        CompoundDevice(lpszFile,lpszDevice){};
        ~Overlay(){};
};

class _EXPORT Scanner: public CompoundDevice
{
public:
        Scanner(LPSTR lpszFile, LPSTR lpszDevice):
                        CompoundDevice(lpszFile,lpszDevice){};
        ~Scanner(){};
};
```

biblio.cpp

```
#include <memory.h>
#include <string.h>
#include <stdio.h>
#include <owldefs.h>
#include "biblio.h"
#define   MCI_BUFSIZE 128

/****************** Funciones MCIDevice ************************/

MCIDevice::MCIDevice(LPSTR lpszDevice){
    wDeviceID = NULL;
    hWndParent = NULL;
    dwErrState = NULL;
    lpszDeviceType = NULL;
    if (lpszDevice)
        SetDeviceType(lpszDevice);
}

MCIDevice::~MCIDevice(){
    if (wDeviceID)
        Close();
    if (lpszDeviceType)
        delete[] lpszDeviceType;
}

LPSTR MCIDevice::Info(DWORD dwFlags){
MCI_INFO_PARMS mciInfoParms;
static char cBuf[MCI_BUFSIZE];
    if (!wDeviceID)
        return (LPSTR) NULL;
}
```

```
    mciInfoParms.lpstrReturn = cBuf; // Apuntador a informacion
    mciInfoParms.dwRetSize = MCI_BUFSIZE; // Tamño del buffer
    dwErrState = mciSendCommand(wDeviceID, MCI_INFO, dwFlags,
                        (DWORD) (LPMCI_INFO_PARMS)&mciInfoParms);
    return mciInfoParms.lpstrReturn;
}


DWORD MCIDevice::Set(DWORD dwFlags, DWORD dwExtra){
MCI_SET_PARMS mciSetParms;
    if (!wDeviceID)
        return MCI_ERR_NOT_OPEN;
    if (dwFlags& MCI_SET_TIME_FORMAT)
        mciSetParms.dwTimeFormat = dwExtra;
    dwErrState = mciSendCommand(wDeviceID, MCI_SET, dwFlags,
        (DWORD)(LPMCI_SET_PARMS)&mciSetParms);
    return dwErrState;
}


DWORD MCIDevice::Status(DWORD dwFlags, DWORD dwItem, DWORD dwExtra){
MCI_STATUS_PARMS mciStatusParms;
    if (!wDeviceID)
        return MCI_ERR_NOT_OPEN;
    mciStatusParms.dwItem = dwItem;
    // Determina campos extra se pondran como banderas
    if (dwFlags & MCI_TRACK)
        mciStatusParms.dwTrack = dwExtra; // Tamano o numero de Tracks
    dwErrState = mciSendCommand(wDeviceID, MCI_STATUS,dwFlags,
        (DWORD)(LPMCI_STATUS_PARMS) &mciStatusParms);
    return mciStatusParms.dwReturn;
}
```

```
DWORD MCIDevice::GetDevCaps(DWORD dwItem){
MCI_GETDEVCAPS_PARMS mciGetDevCapsParms;
    if (!wDeviceID)
        return MCI_ERR_NOT_OPEN;
    mciGetDevCapsParms.dwItem = dwItem;
    dwErrState = mciSendCommand(wDeviceID, MCI_GETDEVCAPS,
            MCI_GETDEVCAPS_ITEM,
            (DWORD)(LPMCI_GETDEVCAPS_PARMS) &mciGetDevCapsParms);
    return mciGetDevCapsParms.dwReturn;
}


void MCIDevice::SetDeviceType(LPSTR lpszDevice){
    if (lpszDeviceType)
        delete [] lpszDeviceType;
    lpszDeviceType = new _far char[lstrlen(lpszDevice) + 1];
    lstrcpy (lpszDeviceType, lpszDevice);
}


void MCIDevice::ErrorMessageBox(){
char szErrorBuf[MAXERRORLENGTH];
    if (mciGetErrorString(dwErrState, (LPSTR)szErrorBuf, MAXERRORLENGTH))
        MessageBox(hWndParent, szErrorBuf, "MCI Error",
                                    MB_ICONEXCLAMATION);
    else
        MessageBox(hWndParent, "Error de MCI desconocido", "MCI Error",
                                    MB_ICONEXCLAMATION);
}
```

```
DWORD MCIDevice::Open(LPSTR lpszDevice /* NULL */ ){
MCI_OPEN_PARMS mciOpenParms;
    if (wDeviceID) //Si esta abierto manda un warning
            return MCI_WARN_ALREADY_OPEN;
    if (lpszDevice) /* Guarda el nombre en lpszDeviceType */
            SetDeviceType(lpszDevice);
    else
            if (!lpszDeviceType)
                    return MCI_ERR_NO_DEVICENAME;
    mciOpenParms.lpstrDeviceType = lpszDeviceType;
    dwErrState = mciSendCommand(NULL, MCI_OPEN,MCI_OPEN_TYPE,
            (DWORD)(LPMCI_OPEN_PARMS) &mciOpenParms);
    if (!dwErrState) //Si se abre correctamente se pone el ID
            wDeviceID = mciOpenParms.wDeviceID;
    return dwErrState;
}

DWORD MCIDevice::Close(){
MMCI_GENERIC_PARMS mciGenericParms;
    if (!wDeviceID)
            return MCI_ERR_NOT_OPEN;
    if (!(dwErrState = mciSendCommand(wDeviceID, MCI_CLOSE, MCI_WAIT,
            (DWORD)(LPMCI_GENERIC_PARMS) &mciGenericParms)));
    wDeviceID = NULL;
    return dwErrState;
}

DWORD MCIDevice::Stop()
{
MCI_GENERIC_PARMS mciGenericParms;
    if (!wDeviceID)
            return MCI_ERR_NOT_OPEN;
    return dwErrState = mciSendCommand(wDeviceID, MCI_STOP, MCI_WAIT,
            (DWORD)(LPMCI_GENERIC_PARMS) &mciGenericParms);
}
```

```
DWORD MCIDevice::Pause(){
MCI_GENERIC_PARMS mciGenericParms;
    if (!wDeviceID)
            return MCI_ERR_NOT_OPEN;
    return dwErrState = mciSendCommand(wDeviceID,
                MCI_PAUSE, MCI_WAIT,
                (DWORD)(LPMCI_GENERIC_PARMS) &mciGenericParms);
}


DWORD MCIDevice::Resume(){
MCI_GENERIC_PARMS mciGenericParms;
    if (!wDeviceID)
                return MCI_ERR_NOT_OPEN;
    return dwErrState = mciSendCommand(wDeviceID,
            MCI_RESUME,MCI_WAIT,
            (DWORD)(LPMCI_GENERIC_PARMS) &mciGenericParms);
}


DWORD MCIDevice::Play(LONG lFrom, LONG lTo){
MCI_PLAY_PARMS mciPlayParms;
DWORD dwFlags =0L;
    if (!wDeviceID)
            return MCI_ERR_NOT_OPEN;
    if (hWndParent) {
            mciPlayParms.dwCallback=(DWORD)(LPVOID)hWndParent;
            dwFlags |= MCI_NOTIFY;
    }
    if (lFrom != current) {
            mciPlayParms.dwFrom = lFrom;
            dwFlags |= MCI_FROM;
    }
```

```
    if (lTo != end) {
        mciPlayParms.dwTo = lTo;
        dwFlags |= MCI_TO;
    }
    dwErrState = mciSendCommand(wDeviceID, MCI_PLAY, dwFlags,
        (DWORD)(LPMCI_PLAY_PARMS) &mciPlayParms);
    return(dwErrState);
}


DWORD MCIDevice::Seek(LONG lTo)
{
MCI_SEEK_PARMS mciSeekParms;
DWORD dwFlags = 0L;
    if (!wDeviceID)
        return MCIERR_NOT_OPEN;
    switch(lTo) {
        case start:
                dwFlags = MCI_SEEK_TO_START;
                    break;
        case end:
                dwFlags = MCI_SEEK_TO_END;
                    break;
        default:
                mciSeekParms.dwTo = (DWORD)lTo;
                dwFlags = MCI_TO;
    }
    dwErrState = mciSendCommand(wDeviceID,
                  MCI_SEEK,dwFlags,
                  (LONG)(LPMCI_SEEK_PARMS) & mciSeekParms);
    return dwErrState;
}
```

```
DWORD MCIDevice::StopAll(){
    return mciSendCommand(MCI_ALL_DEVICE_ID, MCI_STOP, 0, NULL );
}

/****************** Funciones de dispositivos compuestos ********************/

CompoundDevice::CompoundDevice(LPSTR lpszFile, LPSTR lpszDevice)
                                    : MCIDevice(lpszDevice){
    lpszFileName = NULL;
    if (lpszFile)
    Open(lpszFile);
}


CompoundDevice::~CompoundDevice(){
    if (wDeviceID)
    Close();
    if (lpszFileName)
    delete [] lpszFileName;
}


DWORD CompoundDevice::Open(LPSTR lpszFile){
MCI_OPEN_PARMS mciOpenParms;
DWORD dwFlags = 0L;
    if (wDeviceID)
        return MCI_WARN_ALREADY_OPEN;
    lpszFileName = new _far char[lstrlen(lpszFile)+1];
        lstrcpy(lpszFileName, lpszFile);
    mciOpenParms.lpstrElementName = lpszFileName;
    mciOpenParms.lpstrDeviceType = lpszDeviceType;
    dwFlags = MCI_OPEN_ELEMENT | MCI_OPEN_TYPE;
```

```
    dwErrState = mciSendCommand(NULL, MCI_OPEN,dwFlags,
    (DWORD)(LPMCI_OPEN_PARMS) &mciOpenParms);
    if (!dwErrState)
        wDeviceID = mciOpenParms.wDeviceID;
    return dwErrState;
}

DWORD
CompoundDevice::Close(){
    MCIDevice::Close();
    if (lpszFileName) {
        delete [] lpszFileName;
        lpszFileName = NULL;
    }
    return dwErrState;
}

/******************** Funciones AVI ******************************/

DWORD AVI::Position(){
    return Status(MCI_STATUS_ITEM,MCI_STATUS_POSITION,0L);
}

DWORD AVI::Update(HDC hdc){
MCI_DGV_UPDATE_PARMS mciUpdateParms;
DWORD dwFlags;
    if (!wDeviceID)
        return MCIERR_NOT_OPEN;
    mciUpdateParms.hDC = hdc;
    dwFlags = MCI_DGV_UPDATE_HDC | MCI_DGV_UPDATE_PAINT;
        dwErrState=mciSendCommand(wDeviceID, MCI_UPDATE, dwFlags,
        (LONG)(LPMCI_DGV_UPDATE_PARMS) &mciUpdateParms);
    return dwErrState;
}
```

```
DWORD AVI::PutDestination(RECT &rect){
MCI_DGV_PUT_PARMS mciPutParms;
DWORD dwFlags;
    if (!wDeviceID)
        return MCI_ERR_NOT_OPEN;
    mciPutParms.rc = rect;
    dwFlags = MCI_DGV_PUT_DESTINATION | MCI_DGV_RECT;
    dwErrState = mciSendCommand(wDeviceID, MCI_PUT, dwFlags,
                    (LONG)(LPMCI_DGV_PUT_PARMS) &mciPutParms);
    return dwErrState;
}

DWORD AVI::Signal(DWORD dwPosition){
MCI_DGV_SIGNAL_PARMS mciSignalParms;
DWORD dwFlags;
    if (!wDeviceID)
        return MCI_ERR_NOT_OPEN;
    if (!hWndParent)
        return MCI_ERR_NO_PARENT;
    mciSignalParms.dwCallback = (DWORD)(LPVOID)hWndParent;
    mciSignalParms.dwPosition = dwPosition;
    mciSignalParms.dwUserParm = (DWORD)(LPVOID)this;
    dwFlags = MCI_DGV_SIGNAL_AT | MCI_DGV_SIGNAL_USERVAL;
    dwErrState = mciSendCommand(wDeviceID, MCI_SIGNAL, dwFlags,
        (LONG)(LPMCI_DGV_SIGNAL_PARMS) &mciSignalParms);
    return dwErrState;
}

DWORD AVI::Configure(){
MCI_GENERIC_PARMS mciGenericParms;
    if (!wDeviceID)
        return MCI_ERR_NOT_OPEN;
    return dwErrState = mciSendCommand(wDeviceID, MCI_CONFIGURE, 0,
        (DWORD)(LPMCI_GENERIC_PARMS) &mciGenericParms);
}
```

```
DWORD AVI::Cue(DWORD dwTo){
MCI_DGV_CUE_PARMS mciCueParms;
DWORD dwFlags;
    if (!wDeviceID)
        return MCI_ERR_NOT_OPEN;
    mciCueParms.dwTo = dwTo;
    dwFlags = MCI_DGV_CUE_OUTPUT | MCI_TO;
    dwErrState = mciSendCommand(wDeviceID, MCI_CUE, dwFlags,
        (LONG)(LPMCI_DGV_CUE_PARMS) &mciCueParms);
    return dwErrState;}


DWORD AVI::Step(DWORD dwFrames, BOOL bReverse){
MCI_DGV_STEP_PARMS mciStepParms;
DWORD dwFlags;
    if (!wDeviceID)
        return MCI_ERR_NOT_OPEN;
    mciStepParms.dwFrames = dwFrames;
    dwFlags = MCI_DGV_STEP_FRAMES;
    if (bReverse)
        dwFlags |= MCI_DGV_STEP_REVERSE;
    dwErrState = mciSendCommand(wDeviceID, MCI_STEP, dwFlags,
        (LONG)(LPMCI_DGV_STEP_PARMS) &mciStepParms);
    return dwErrState;
}


DWORD AVI::SetAudioVolume(DWORD dwVolume){
MCI_DGV_SETAUDIO_PARMS mciSetAudioParms;
DWORD dwFlags;
    if (!wDeviceID)
        return MCI_ERR_NOT_OPEN;
    mciSetAudioParms.dwValue = dwVolume;
    mciSetAudioParms.dwItem = MCI_DGV_SETAUDIO_VOLUME;
```

```
    dwFlags = MCI_DGV_SETAUDIO_ITEM | MCI_DGV_SETAUDIO_VALUE;
    dwErrState = mciSendCommand(wDeviceID, MCI_SETAUDIO, dwFlags,
            (LONG)(LPMCI_DGV_SETAUDIO_PARMS) &mciSetAudioParms);
    return dwErrState;
}


DWORD AVI::SetSpeed(DWORD dwSpeed){
MCI_DGV_SET_PARMS mciSetParms;
DWORD dwFlags;
    if (!wDeviceID)
            return MCI_ERR_NOT_OPEN;
    mciSetParms.dwSpeed = dwSpeed;
    dwFlags = MCI_DGV_SET_SPEED;
    dwErrState = mciSendCommand(wDeviceID, MCI_SET, dwFlags,
            (LONG)(LPMCI_DGV_SET_PARMS) &mciSetParms);
    return dwErrState;
};


DWORD AVI::Window(HWND hWnd){
MCI_DGV_WINDOW_PARMS mciWindowParms;
DWORD dwFlags;
    if (!wDeviceID)
            return MCI_ERR_NOT_OPEN;
    mciWindowParms.hWnd = hWnd;
    dwFlags = MCI_DGV_WINDOW_HWND;
    dwErrState = mciSendCommand(wDeviceID, MCI_WINDOW, dwFlags,
            (LONG)(LPMCI_DGV_WINDOW_PARMS) &mciWindowParms);
    return dwErrState;
};
```

```
/****************** Funciones de Wave ********************/
DWORD Wave::Channels(){
    return Status(MCI_STATUS_ITEM, MCI_WAVE_STATUS_CHANNELS,0L);
}


DWORD Wave::BitsPerSample(){
    return Status(MCI_STATUS_ITEM, MCI_WAVE_STATUS_BITSPERSAMPLE,0L);
}


DWORD Wave::SamplesPerSecond(){
    return Status(MCI_STATUS_ITEM, MCI_WAVE_STATUS_SAMPLESPERSEC,0L);
}


DWORD Animation::Window(HWND hWnd){
MCI_ANIM_WINDOW_PARMS mciWindowParms;
DWORD dwFlags=0L;
    if (!wDeviceID)
        return MCI_ERR_NOT_OPEN;
    mciWindowParms.hWnd = hWnd;
    dwFlags |= MCI_ANIM_WINDOW_;
    dwErrState = mciSendCommand(wDeviceID, MCI_WINDOW, dwFlags,
        (LONG)(LPMCI_ANIM_WINDOW_PARMS) &mciWindowParms);
    return dwErrState;
};
```

Bibliografía

[1] Sick, Gary
ObjectWindows how-to:the definitive Borland C++ problem solver for Windows
Waite Group
Corte Madera, California
1993

[2] Rimmer, Steve
Multimedia Programming for Windows
Mc-GrawHill
New York
1994

[3] Arch C. Luther
Designing Interactive Multimedia
Bantam
New York
1992

[4] Microsoft Corporation
Multimedia Programmer's Reference
1992

[5] Microsoft Corporation
Multimedia Programmer's Guide
1992

[6] Stroustrup, Bjarne
The C++ Programming Language
Addison Wesley
Massachusetts
1987

[7] Borland
Turbo C++ 3.0 for Windows. Programmer's Guide. Versión 3.0
1991

[8] Borland
ObjectWindows for C++. User's Guide
1993