Interfaz telefónico de acceso remoto para una aplicación de control domótico

Departamento de Ingeniería de Sistemas y Automática

Escuela Superior de Ingenieros

Universidad de Sevilla

Autor:

Jesús Carranza Tascón

Tutor:

D. Eduardo Fernández Camacho

Mi agradecimiento a todas esas personas que han estado a mi lado apoyándome mientras realizaba este Proyecto Fin de Carrera.

En especial mis gracias a D. Eduardo Fernández Camacho, Catedrático del Departamento de Ingeniería de Sistemas y Automática, sin cuya ayuda este Proyecto Fin de Carrera no habría sido posible.

INDICE:

INDICE DE ILUSTRACIONES:

Interfaz telefónico de acceso remoto para una aplicación de control domótico

1 Descripción del problema

Gracias a los modernos sistemas informáticos y al abaratamiento de los mismos, actualmente es relativamente fácil de instalar y mantener un sistema de control domótico en una vivienda media. Estos sistemas se basan en ordenadores personales, tipo PC, conectados mediante tarjetas de entrada y salida a los distintos sensores y actuadores del sistema de control. El coste de este conjunto de elementos hardware requerido para la instalación de un sistema de control es relativamente bajo, lo que puede favorecer la sucesiva implantación de estos sistemas y su generalización en un mercado futuro. Si ha este relativamente bajo coste de instalación se le añade el actual sistema programación en entornos visuales, con interfaces basados en ventanas, faciles de usar y compreder aún por usuarios no versados en sistemas de control, tenemos una mas que favorable situación para el desarrollo de estos sistemas.

Por otro lado el actual aumento de los sistemas de comunicaciones ha favorecido la implantación de todo un conjunto de servicios basados en la comunicación a distancia. La telefonía móvil, por ejemplo, ha generado una mentalidad en la sociedad actual que premia y busca la posibilidad de actuar sobre aquello que se desea en cualquier momento y situación. Así han aparecido servicios tales como sistemas de compra desde teléfonos móviles, Internet para comunicaciones móviles (WAP), etc... Del mismo modo el actual Internet permite todo tipo de transacciones económicas a distancia, desde la consulta del saldo de una cuenta bancaria hasta la reserva de una plaza de hotel en una ciudad situada al otro extremo del mundo. Es mas, la actual tendencia de las dadas a llamar tecnologías de la información, hace pensar en un mayor reforzamiento de esta situación.

Por todo ello cabe plantearse la posibilidad de proporcionar a los actuales sistemas de control domótico de un sistema de teleoperación o teleacceso. Dado que el sistema se basa en un ordenador instalado en la vivienda el modo de acceso a las aplicaciones de control de dicho sistema se realizaría mediante el teclado, el ratón y el monitor del ordenador. Esto obliga al usuario a permanecer necesariamente en la vivienda si desea actuar sobre el sistema o

comprobar algún estado o valor del mismo. No solo en estos casos el usuario estaría restringido en su modo de actuar, sino que en caso de producirse alguna situación de emergencia o especialmente significativa en la que fuese conveniente informar al usuario, sería del todo imposible si este no estuviese en la vivienda. Es por todo ello que parece más que útil el dotar a los sistemas domóticos de algún tipo de interfaz con un sistema de comunicaciones capaz de prolongarle hasta la ubicación del usuario, es decir, que prolongue el interfaz con la aplicación de gestión del sistema domótico más halla de la ubicación física del ordenador en el que está instalada. Es en este campo en el que nos centraremos en este proyecto fin de carrera: **Proporcionar un interfaz con el usuario del sistema de control domótico que le permita acceder al mismo desde cualquier ubicación**.

2 Soluciones al problema planteado

Considerando el problema planteado se barajaron varías soluciones basadas cada una de ellas en una tecnología distinta. Una vez establecidas las mas adecuadas se plantearon las ventajas e inconvenientes de cada una de ellas con el fin de poder establecer un criterio firme en el que basarnos para seleccionar la que se emplearía en el presente trabajo.

Así a la vista del auge que actualmente presentan aquellas tecnologías basadas en comunicaciones móviles se decidió seleccionar aquellas que fueran compatibles con esta tecnología. Es decir, la tecnología que se emplease para desarrollar nuestro interfaz debería ser tal que fuese accesible desde un teléfono móvil. Esto nos garantizaría que su acceso fuese del todo independiente de la localización física del usuario, siempre y cuando la comunicación desde y hacia dicho teléfono móvil esté disponible.

Una segunda condición que debería cumplir nuestro sistema de acceso remoto es que su coste no debería ser significativo en comparación con el resto del sistema. Esto garantizaría un aumento de la funcionalidad del sistema de control domótico sin un aumento significativo de su coste, lo que aumentará su atractivo y facilitará su implantación. Por lo tanto la tecnología que se emplease en el desarrollo de nuestro interfaz no debería requerir de ni equipos

costosos ni de unas instalaciones especialmente complicadas, debiendo de poder ser accesibles por la gran mayoría.

Finalmente el sistema desarrollado debería cumplir una tercera y última condición. Debe tenerse en cuenta que el usuario final del sistema de control domótico será, en la mayoría de los casos, alguna persona no necesariamente conocedora de las tecnologías empleadas en su desarrollo. Por lo tanto debería de establecerse como condición para el desarrollo de nuestro interfaz la facilidad de instalación y mantenimiento. Del mismo modo, el interfaz debe resultar cómodo e intuitivo para dicho usuario final, sin que su uso requiera de unas instrucciones o procesos complejos.

Planteando como condiciones a cumplir por nuestro interfaz las tres premisas anteriormente citadas, se conseguiría un interfaz que, sin aumentar considerablemente el coste del sistema de control domótico, ofreciese la funcionalidad de este, como sistema de control, extendiéndola hasta la localización del usuario del sistema independientemente de su ubicación, fácilmente y sin un gran esfuerzo por parte del usuario. Esto aumentaría sin duda el atractivo de estas aplicaciones, extendiendo de forma significativa su funcionalidad.

Teniendo en cuanta esta tres condiciones se barajaron tres opciones:

- Emplear un servidor WAP conectado a nuestra aplicación de control.
- Emplear un MODEM GSM conectado al PC del sistema de control domótico para el envío y recepción de mensajes SMS.
- Emplear un MODEM Analógico conectado al PC para la realización y recepción de llamadas de voz.

La primera de ellas se basa en el desarrollo de un servicio WAP que proporcionaría las opciones de actuación sobre el sistema. La pagina WAP visualizada en la pantalla del teléfono móvil del usuario, una vez que este se hubiese conectado al correspondiente servidor, mostraría un conjunto de menús con opciones seleccionables. El usuario seleccionaría una de las opciones tras lo cual el servidor WAP comunicaría a la aplicación de control domótico la correspondiente actuación a realizar sobre el sistema.

La segunda opción se basa en la recepción y envío de mensajes SMS con un formato preestablecido para actuar sobre el sistema. Para ello se sería necesario conectar el PC del sistema de control domótico a la red de alguno de los operadores de telefonía móvil mediante un MODEM GSM. De este modo el usuario enviaría mediante su teléfono móvil un mensaje SMS, con el formato establecido, al número de teléfono asignado al PC del sistema de control domótico. El sistema interpretaría el mensaje recibido y comunicaría la correspondiente actuación al sistema de control domótico.

La última opción se basa en la recepción de llamadas de voz mediante la línea telefónica analógica estandar. Para ello se conectaría el PC del sistema de control domótico a la línea telefónica mediante un MODEM Analógico con funciones de voz, lo que se conoce como un VoiceModem. El usuario realizaría una llamada al número telefónico asignado a la línea a la que se ha conectado el PC. La aplicación respondería con un mensaje de voz en el que se ofrecen una serie de opciones accesibles para el usuario mediante la pulsación de una de las teclas de su teléfono móvil. El sistema interpretaría la pulsación de la tecla y comunicaría la correspondiente actuación al sistema de control domótico.

3 Solución seleccionada

Teniendo en cuenta los requisitos que el interfaz debe cumplir y por los motivos que más adelante se detallarán, se decidió realizar la última de las tres soluciones planteadas. Es por ello que en el presente proyecto fin de carrera se propone el desarrollo de una herramienta que permita la conexión de un supuesto sistema de control domótico con la línea telefónica analógica, permitiendo al sistema de control la recepción y realización de llamadas desde y hacia el exterior.

La decisión se fundamenta en varios motivos:

- Bajo coste:

Es el principal motivo por el que se ha seleccionado esta opción. Los elementos tanto hardware como software necesarios para dar soporte a esta solución están ya presentes en la

mayoría de los hogares actuales. No solo eso, sino que en caso de no tenerse ya instalados en la vivienda el coste de instalación es muy bajo.

El soporte hardware necesario para la instalación del sistema de acceso planteado se basa en una línea telefónica analógica fija estandar, en un MODEM analógico y un ordenador personal tipo PC.

La telefonía fija es algo presente prácticamente en la totalidad de los hogares actuales, por lo que en caso de querer instalarse un sistema de control domótico en la vivienda con nuestro sistema de acceso remoto, el soporte físico ya estaría instalado, por lo que su contratación sería innecesaria. Esto supondría un ahorro a la hora de realizar la instalación. En caso de no estar disponible la línea telefónica, la contratación de la misma y su posterior mantenimiento serían significativamente más baratos que la contratación y mantenimiento de otros tipos de accesos, como los que ofrecen conexiones permanentes a Internet requeridos por un sistema basado en WAP.

El segundo elemento hardware requerido por nuestro interfaz de acceso remoto es un MODEM analógico con funciones de voz, que nos permita la conexión del PC sobre el que se instalará la aplicación de control domótico con la línea telefónica anteriormente comentada. Dicho MODEM debe soportar funciones de voz para la realización y recepción de este tipo de llamadas. Este tipo de MODEM se ha estandarizado por la disminución de los precios de este tipo de dispositivos, por lo que casi la totalidad de los MODEMs analógicos actuales ofrecen estas funciones. No solo se ha generalizado este tipo de dispositivos, sino que además, y gracias al auge que han tenido en los últimos años los servicios de acceso a Internet, se han generalizado en los ordenadores personales, estando ya instalados por defecto en la mayoría de este tipo de equipos informáticos. Es por ello que en una gran mayoría de los casos la vivienda en la que se vaya a instalar el sistema de control domótico contará ya con la presencia de un ordenador personal con un MODEM analógico. En caso de no disponerse de este tipo de dispositivo, su coste es mínimo, rondando los 80 o 100 euros, significativamente más económicos que los correspondientes MODEM GSM requeridos por el sistema basado en mensajes SMS, que suelen rondar los 350 o 400 euros.

Finalmente el último de los elementos hardware requeridos por nuestro sistema de control domótico se halla igualmente generalizado, ya que en los últimos años este tipo de equipos han bajado su coste y aumentado sus prestaciones y versatilidad. Del mismo modo que los MODEM analógicos una mayoría de las viviendas sobre las que se instale nuestro sistema de control domótico contará ya con un ordenador personal, por lo que se ahorrará el coste de adquisición del mismo.

Por otro lado el soporte software para nuestro sistema control domótico se basará en un sistema operativo tipo Windows. La selección de este tipo de sistema operativo se basa en dos consideraciones: Ofrece un entorno amigable e intuitivo basado en ventanas en el que se pueden desarrollar aplicaciones complejas que sin embargo ofrecen al usuario un interfaz fácil de usar y entender; y se haya instalado en la gran mayoría de los ordenadores personales, por lo que no se haría necesario la adquisición e instalación de nuevos sistemas operativos. Esto nos lleva a considerar que el empleo de este sistema operativo evitaría aumentar la complejidad del sistema, el usuario seguiría usando un sistema operativo al que ya esta acostumbrado sin necesidad de mantener otros sistemas, que en la mayoría de los casos le serán desconocidos, y no aumentaría el coste de la instalación de nuestro sistema de control domótico.

- Facilidad de uso:

El sistema que se ha elegido ofrece una elevada facilidad de uso al tratarse de un interfaz basado en mensajes de voz autoexplicativos, que ofrecen una serie de opciones seleccionables mediante la pulsación de la correspondiente tecla en el teléfono desde el que se realiza la llamada. De este modo el usuario no debe recordar ningún esquema o sistema para el acceso y uso del interfaz, ya que es el propio interfaz el que le conduce a través de las distintas opciones y menús. Del mismo modo el método de acceso al interfaz es sencillo y fácil de recordar, ya que se basa en realizar una llamada normal al número de teléfono asignado a la línea telefónica a la que esté conectado el MODEM, que en la mayoría de los casos coincidirá con el teléfono particular de la vivienda.

A esta facilidad de uso se añade la flexibilidad del sistema a la hora de identificar cada uno de los elementos del sistema de control domótico. El método empleado para identificar uno de estos elementos, una alarma, un actuador, etc..., se basa en un código numérico de tres cifras que es configurable por el propio usuario. Es decir, no es impuesto por el interfaz, sino que es el usuario el que en podrá determinar la correspodencia entre cada uno de los elementos del sistema de control domótico, que obviamente estarán determinados y serán fijos, y uno o más de estos códigos numéricos. Esto facilitará el uso del interfaz por parte del usuario ya que es él mismo el que determina estas correspondencias.

La misma filosofía se ha seguido a la hora de establecer los avisos y emergencias que el sistema de control domótico notificará al usuario mediante el interfaz telefónico. Así estos avisos y emergencias son configurables por parte del usuario, quien será el responsable de configurar la prioridad de los mismos así como de determinar si el sistema realiza o no la correspondiente notificación en caso de producirse el aviso o emergencia.

Por último la facilidad de uso del sistema se ve reforzada por las funciones de configuración del interfaz de acceso. Así el sistema ofrece al usuario las funciones necesarias para que este configure los parámetros necesarios para el correcto funcionamiento del interfaz. Así ofrece funciones para establecer la clave de acceso que el interfaz solicita para facilitar el acceso a las opciones de actuación sobre el sistema y el número de tonos que el interfaz esperará para tomar el control de la llamad, como parámetros de configuración de las llamadas entrantes. Del mismo modo ofrece funciones para configurar las llamadas salientes, utilizadas por el interfaz en caso de realizar alguna notificación al usuario, ofreciendo la posibilidad de configurar el número de teléfono al que se llamará, pudiendo ser tanto un fijo como un móvil, y el país al que pertenece este número, como parámetro necesario para la realización de la llamada.

Todas estas opciones y funciones estarán envueltas por un entorno basado en ventanas y diálogos que facilita aun más su uso y comprensión.

- Facilidad de mantenimiento:

En este caso el sistema planteado ofrece una doble facilidad: tanto al usuario final de la aplicación como al desarrollador de la misma.

Desde el punto de vista del desarrollador el sistema planteado ofrece una gran facilidad a la hora de su instalación. Al tratarse de una aplicación que engloba toda la funcionalidad requerida por el sistema de control y el interfaz de acceso remoto, no se requiere de la instalación de aplicaciones adicionales. Del mismo modo al tratarse de una aplicación basada en un entorno Windows, el proceso de instalación es susceptible de ser automatizado mediante el empleo de las herramientas que el entorno de desarrollo empleado ofrece, lo que facilitaría aun más el proceso de instalación de la aplicación. Pero la facilidad que ofrece el sistema planteado se extiende a la hora de mantener la aplicación. El interfaz de acceso será desarrollado como un módulo externo a la aplicación base del sistema de control domótico, por lo que la actualización de uno u otro componente puede realizarse sin que se vea afectado el otro, manteniendo un interfaz de comunicaciones mínimo entre ambos.

Desde el punto de vista del usuario final la facilidad de mantenimiento se ve reflejada en el mismo echo de no necesitarse aplicaciones o servicios especiales para su funcionamiento. Esto evita que usuario tenga que acostumbrarse al manejo de aplicaciones al margen de la propia del sistema de control domótico, pudiéndose centrar en aprender y comprender el funcionamiento de ésta última. Esto facilitará que el usuario, en un corto período de tiempo, domine y controle la aplicación de control domótico. Además las conexiones entre los distintos dispositivos que dan soporte al interfaz son sencillas y fáciles de realizar y mantener, conectar el MODEM a la línea telefónica y al PC; y el mantenimiento de los mismos puede hacerse desde el propio sistema operativo, sin un especial esfuerzo.

4 Inconvenientes de la solución adoptada

El sistema planteado ofrece un inconveniente en el método seleccionado para la comunicación con el usuario. Ésta se basa en un conjunto de mensajes de voz pregrabados necesarios para reproducir tanto los menús que forman el interfaz de actuación sobre el

sistema como los mensajes de información y aviso que se enviarán al usuario. Este sistema obliga a mantener estos mensajes en ficheros de voz que serán cargados y reproducidos, lo que supondría un aumento de la complejidad de instalación y mantenimiento de la herramienta.

Para evitar este inconveniente se ha optado por encapsular todos los mensajes de voz necesarios para el correcto funcionamiento del interfaz telefónico, menús y mensajes necesarios para reproducir el estado y valor de los elementos del sistema de control domótico, en un módulo adicional. De este modo los mensajes de voz resultan totalmente transparentes para el módulo de control del interfaz y para el usuario final. Además de este modo se garantiza que durante el desarrollo de la aplicación de control domótico puedan ser alterados sin que esto afecte al funcionamiento del interfaz de acceso remoto, con la única condición de mantener el interfaz establecido entre ambos módulos.

Sin embargo, y a pesar de esta solución, sigue existiendo un inconveniente a la hora de reproducir los mensajes de voz correspondientes a los mensajes de información y avisos. Estos dependen del elemento del sistema de control domótico sobre el que se actúe, por lo que no pueden ser considerados como una parte del interfaz telefónico si se quiere garantizar la correcta estructuración del sistema en módulos totalmente independientes. Para resolver este punto se puede optar entre dos soluciones:

- Pregrabar los mensajes necesarios para la reproducción de las peticiones de información y avisos y encapsularlos en un módulo adicional. De este modo es durante el desarrollo de la aplicación de control domótico cuando se establecerían dichos mensajes, quedando totalmente enmascarados para el usuario final.
- Pregrabar los mensajes necesarios pero no encapsularlos, dejando al usuario libertad para modificarlos y alterarlos. De este modo el usuario podrá ajustar esos mensajes a su propio gusto.

En el presente proyecto fin de carrera se ha optado por la segunda opción, ya que ofrece una mayor flexibilidad al usuario final.

Existe una última posibilidad para la reproducción de los mensajes de voz basada en el empleo de motores de conversión de texto a voz, motores TTS, capaces de convertir un texto escrito a una señal de voz y redirigirla hacia el MODEM. En la actualidad existen librerías de libre distribución, como SAPI de Microsoft, que ofrecen todo un conjunto de componentes y funciones para el desarrollo y empleo de motores TTS y RS, éstos últimos para realizar el proceso inverso y que son la base de los sistemas de reconocimiento de voz. Mediante el empleo de estas herramientas podría proporcionarse al interfaz de acceso una mayor flexibilidad a la hora de reproducir los mensajes de voz. Para más detalles consúltese el Anexo 1 del presente proyecto fin de carrera[1].

5 Descripción funcional

Establecidas y definidas las características a las que el interfaz de acceso remoto a la aplicación de control domótico debe de dar soporte, debemos establecer de una forma clara y concisa la funcionalidad que se espera del mismo. De este modo estableceremos los requisitos derivados tanto de estas características como de la funcionalidad que se espera del mismo, lo que nos facilitará el desarrollo y comprensión de la herramienta.

Como primer requisito a cumplir por la herramienta desarrollada cabe mencionarse la modularidad de la misma, con el fin de garantizar su reutilización por otras aplicaciones, tanto desarrolladas para este mismo campo, la domótica, como para todas aquellas que requieran de un sistema de acceso remoto. Esta modularidad e independencia respecto de la aplicación base que la utilice garantiza además su futura actualización y modificación, sin necesidad de realizar profundas adaptaciones y modificaciones en dicha aplicación base. Esta condición determinará el formato seleccionado para el desarrollo de la herramienta, que como se verá en el apartado siguiente se ha encapsulado en un módulo reutilizable.

Desde el punto de vista funcional, la herramienta básicamente deberá ofrecer las funciones y objetos necesarios para la recepción, realización y control de llamadas telefónicas. Se trata pues no solo de funciones permitan la realización de dichas llamadas, sino que además deberán proporcionar todo un conjunto de mecanismos para el control de las mismas,

[1] Página 51

entendiendo por control la capacidad de detectar las llamadas que se reciban, aceptarlas y finalizarlas, así como todo el conjunto de funciones necesarias para la configuración de las llamadas y del MODEM empleado.

Por lo tanto la funcionalidad de la herramienta puede desglosarse en tres apartados:

5.1 Configuración del interfaz

En este apartado el interfaz ofrece a la aplicación base los métodos necesarios para su configuración, tanto de aquellos que afecten al interfaz en su totalidad como de aquellos que influyan sobre la recepción y realización de llamadas.

De este modo el interfaz ofrece la funcionalidad necesaria de detectar y listar los MODEM compatibles con el tipo de las llamadas que se recibirán y que se realizarán. Estas llamadas son, como ya se ha comentado, llamadas de voz, por lo que el MODEM que se seleccione como soporte para el interfaz debe de dar soporte a este tipo de llamadas. Para evitar posibles incompatibilidades, el interfaz ofrece esta funcionalidad con vistas a una posible selección por parte del usuario final del MODEM que desee emplear, en caso de tener más de uno instalado. Por otro lado, y continuando con las características generales configurables, el interfaz ofrece la funcionalidad para establecer el MODEM sobre el cual se instalará, es decir sobre el cual monitorizará la línea telefónica, recibirá llamadas y realizará llamadas. Este parámetro de la configuración del interfaz no puede ser modificado en tiempo de ejecución, es decir si se ha abierto ya una línea sobre un MODEM, en cuyo caso primero se debería de cerrar la ya abierta para posteriormente abrir una nueva. Para esto último el interfaz ofrece funciones de conexión, desconexión, arranque y parada, así como funciones de chequeo del estado del interfaz, proporcionando de esta forma a la aplicación base un control total sobre el estado del interfaz y su configuración. De esta forma el interfaz de acceso remoto puede habilitarse o deshabilitarse desde la aplicación base de forma que no necesariamente estará disponible, pudiéndose habilitarse o deshabilitarse según el criterio del usuario.

Como herramientas adicionales para la configuración de los parámetros relacionados con la realización y recepción de llamadas, el sistema ofrece funciones de acceso a las configuración tanto del MODEM seleccionado como de las propiedades de marcado asociadas a dicho MODEM. Estas funciones, junto con las que ofrece el propio sistema operativo, permiten al usuario controlar con profundidad las características del MODEM empleado para dar soporte al interfaz.

Desde el punto de vista de la recepción de llamadas los parámetros configurables que determinan el funcionamiento del interfaz son dos: el código numérico que constituye la clave de acceso al sistema y el número de tonos que esperará el interfaz antes de tomar el control de la llamada entrante. Para poder establecer y configurar dichos valores el interfaz ofrece las correspondientes funciones. De este modo el usuario podrá determinar según su propio criterio la clave de acceso al sistema, lo que ofrece un nivel de seguridad considerablemente alto. Este nivel de seguridad se requiere ya que al tratarse de un interfaz fácilmente accesible desde el exterior, basta una llamada telefónica al número asociado a la línea conectada al MODEM, se debe proteger las opciones de actuación sobre el sistema de control con algún sistema de autenticación del usuario. Por otro lado se puede configurar el número de tonos que esperara el interfaz antes de tomar el control de la llamada entrante.

Por último para la realización de llamadas los parámetros que pueden configurarse son el número de teléfono al cual se llamará y el país al que pertenece dicho número. El conjunto de ambos valores determinará el formato definitivo del número telefónico que se marcará al realizar la llamada. El sistema empleará el número del país según la configuración de las propiedades del MODEM empleado para realizar las llamadas. Por lo tanto el sistema ofrece total libertad para determinar el número de teléfono al que se llamará, en caso de tener que realizar una llamada, por lo que el usuario podrá ajustar este parámetro según su propia conveniencia.

5.2 Recepción de llamadas y acceso al sistema

Como ya se ha comentado la funcionalidad que ofrece el interfaz para el acceso al sistema de control domótico se basa en un conjunto de menús autoexplicativos reproducidos a través de la línea telefónica. Dichos menús ofrecen una serie de opciones que el usuario podrá seleccionar pulsando sobre la correspondiente tecla del teléfono desde el que realiza la llamada. De este modo la funcionalidad ofrecida por el interfaz puede agruparse según cada uno de estos menús.

El sistema de menús planteado añade una funcionalidad adicional con el fin de evitar que una llamada recibida mantenga la línea ocupada sin tener ningún tipo de actividad o a pesar de haberse finalizado por parte del llamante. De este modo cada vez que se finaliza la reproducción de uno de los mensajes el sistema permanece a la escucha para detectar la pulsación de una de las teclas. Si transcurrido un período de tiempo sin haber detectado actividad alguna en la línea telefónica, el interfaz da por finalizada la llamada, colgando y dejando nuevamente en reposo la línea telefónica. Este funcionamiento se implementa con el fin preventivo de evitar llamadas malintencionadas, perdidas de la comunicación o períodos de inactividad en la llamada demasiado largos, que dejen al sistema incomunicado al tener la línea ocupada.

5.2.1 Menú de Inicio

Forma el primer menú que se presenta al usuario tras la recepción de la llamada. El interfaz detecta la llamada entrante a través de la línea a la que se ha conectado, informando de ello a la aplicación base tal y como se explicará más adelante. Tras esperar el número de tonos especificado tomará el control de la llamada, descolgará y reproducirá este primer menú.

En este menú de bienvenida se ofrece una pequeña introducción al sistema, a modo de explicación del mismo, pasándose a ofrecer dos opciones, asociadas a sendas teclas. Así las opciones ofrecidas son: acceder al sistema, pulsando la tecla 1, o salir del sistema, pulsando la tecla 3.

Si la opción seleccionada por el usuario es la de salir del sistema, el interfaz, tras detectar la pulsación de la tecla, procederá a colgar y terminar la llamada, dejando nuevamente en reposo la línea para poder recibir o realizar nuevas llamadas.

Si la opción seleccionada por el usuario es la de acceder al sistema, el interfaz pasará a reproducir el siguiente menú, el **Menú de Acceso**.

5.2.2 Menú de Acceso

Este segundo menú da soporte al mecanismo de validación del usuario y garantiza el nivel de seguridad requerido por el interfaz de acceso al sistema de control domótico. El menú solicita al usuario que introduzca la clave numérica de cinco cifras que forma la llave de acceso al sistema de control y monitoriza la introducción de dicha clave, de forma que si se comete algún error en su introducción o si la clave introducida no coincide con la establecida, tal y como se comentó anteriormente[1], finaliza la comunicación colgando y cerrando la llamada. Si por el contrario la clave introducida coincide con la establecida, el interfaz valida al usuario y pasa al siguiente menú: el **Menú de Opcciones**, desde el cual se podrá ya acceder a las opciones de actuación sobre el sistema de control domótico.

5.2.3 Menú de Opciones

Como ya se ha mencionado este menú ofrece las opciones necesarias para acceder a las funciones de actuación sobre el sistema de control domótico. Así ofrece tres opciones al usuario. La primera de ellas accede al **Menú de Acción** desde el que se podrá actuar sobre alguno de los elementos del sistema de control, asociada a la tecla 1; la segunda accede al **Menú de Información**, desde el cual se podrá solicitar información sobre los elementos del sistema de control, asociada a la tecla 2; y una última opción, asociada a la tecla 3, de finalización de la comunicación.

[1] Véase el apartado 5.1 Configuración del Interfaz

De este modo y según la tecla pulsada por el usuario en el teléfono empleado para la realización de la llamada, el interfaz pasará a reproducir un menú u otro.

5.2.4 Menú de Acción

Este menú ofrece la opción de actuar sobre uno de los elementos del sistema de control domótico. Para ello solicita al usuario la introducción del código numérico de tres cifras que identifica al elemento sobre el que desea actuar seguido del valor numérico que se desea establecer como actuación sobre el sistema. La introducción de ambos valores debe de finalizarse con la pulsación de la tecla '#' para delimitar uno del otro y evitar equívocos. Tras introducir los dos parámetros requeridos para la actuación, el interfaz notifica a la aplicación base de la petición de actuación. La aplicación base realizaría la actuación a partir de los datos proporcionados por el interfaz, los introducidos por el usuario, y comunicaría a este el resultado de la operación. Con esta información el interfaz reproduce un mensaje de información sobre la finalización de la actuación, si se ha realizado correctamente o no, y vuelve al **Menú de Opciones**, dando por finalizada la acción sobre el sistema de control domótico.

El interfaz en este punto controla el formato de la clave numérica que identifica al elemento, generando un mensaje de error en caso de detectar un fallo en la introducción y reseteando la información introducida hasta ese momento. No finaliza la comunicación ya que en este estado se considera que el usuario conectado ha sido validado satisfactoriamente y posee todos los privilegios para acceder al sistema de control domótico.

5.2.5 Menú de Información

Este menú ofrece la opción de pedir información sobre uno de los elementos del sistema de control domótico. Para ello solicita al usuario la introducción del código numérico de tres cifras que identifica al elemento sobre el que desea pedir información seguido la tecla '#,' para delimitar la clave numérica y evitar equívocos. Tras introducir la clave numérica el interfaz notifica a la aplicación base de la petición de información. La aplicación base realizaría la correspondiente actuación a partir de los datos proporcionados por el interfaz, obtendría el

estado actual del elemento que el usuario ha identificado y notificaría al interfaz dicho estado, además de los datos adicionales que este necesitará para generar el mensaje de información. El interfaz reproduce un mensaje de información a partir del estado del elemento seleccionado y vuelve al **Menú de Opciones**, dando por finalizada la petición de información sobre el sistema de control domótico.

El interfaz en este punto controla, al igual que en el menú anterior, el formato de la clave numérica que identifica al elemento, actuando de igual forma ante un error en el formato de la misma.

Alcanzado el **Menú de Opciones** el usuario validado por el interfaz de acceso al sistema de control domótivo podría realizar tantas acciones y/o peticiones de información sobre el sistema ce control como desee, hasta que seleccione la opción de desconexión asociada a la tecla 3 de dicho menú, momento en el que se da por finalizada la comunicación, finalizándose la llamada y dejándose la línea nuevamente en reposo para la recepción o realización de nuevas llamadas.

5.3 Realización de llamadas y envío de avisos

El interfaz de acceso al sistema de control domótico ofrece una funcionalidad extendida para la comunicación remota al usuario de emergencias o avisos detectados por el sistema de control. De este modo el interfaz podrá ser notificado de la existencia de una de estas emergencias o avisos, pasando este a notificárselo al usuario. El mecanismo de notificación de emergencias se basa, al igual que el sistema de actuación sobre el sistema, en mensajes de voz reproducidos a través de la línea telefónica asociada al interfaz, aunque en este punto no se trata de menús u opciones, sino en un mensaje explicativo de la emergencia o aviso mediante el cual el usuario sea capaz de determinar la causa exacta del aviso.

Por lo tanto tras producirse una de las emergencias detectables por el sistema de control domótico, la aplicación base, tras recibir la correspondiente notificación de la emergencia,

procederá a informar al interfaz de la ocurrencia de la misma. Para ello le proporcionará la información necesaria para que este reproduzca el mensaje de voz asociado a la emergencia y la prioridad de la misma. Obtenida esta información el interfaz procederá a registrar la emergencia producida iniciándose el proceso de notificación de avisos.

Dicho proceso consta de la realización de una llamada al número de teléfono configurado en el interfaz[1] y de la reproducción del mensaje de voz indicado al recibir la notificación del aviso. El proceso de notificación controla la recepción del aviso por parte del usuario, ya que espera la pulsación de una de las teclas del teléfono usado por el usuario para la recepción de avisos como mecanismo de confirmación. De este modo si tras la reproducción de un cierto número de veces del mensaje de aviso no se ha obtenido respuesta afirmativa por parte del usuario, tal y como se ha explicado, el interfaz da por finalizada la actual notificación, pero no desregistra la emergencia, quedando esta aún registrada. Pasado un cierto tiempo el interfaz recupera las emergencias registradas en el sistema y, en caso de haber alguna aún no confirmada, pasaría a notificar automáticamente la de mayor prioridad, reiniciándose el proceso de notificación de avisos.

El proceso de registro de las emergencias que lleva a cabo el interfaz de acceso remoto se basa en una clasificación de las mismas según su prioridad. De esta forma cuando el interfaz recibe la notificación de una nueva emergencia la registra según su prioridad, de mayor a menor, tras lo cual pasa a reproducir la emergencia de mayor prioridad, que como puede apreciase no necesariamente coincidirá con la notificada al interfaz. Teniendo en cuenta que el sistema no desregistra las emergencias hasta que no se recibe la correspondiente confirmación de su recepción por parte del usuario, se garantiza que el éste siempre será notificado de las emergencias o avisos de mayor importancia o urgencia. Se evita por lo tanto que avisos poco importantes o de poca relevancia enmascaren a otros de mayor peso, ocupando la línea y impidiendo al sistema la reproducción de los realmente significativos.

6 Descripción técnica

[1] Véase el apartado 5.1 Configuración del Interfaz

Establecida la funcionalidad básica a la que dará soporte nuestro interfaz de acceso remoto desarrollamos todo un conjunto de clases y objetos con el fin de dar soporte a esta funcionalidad. Dichas clases y objetos han sido encapsulados en un fichero de extensión de aplicación, mas conocido como DLL, de forma que todos los recursos por él encapsulados puedan ser empleados por diversas aplicaciones sin que estas últimas tengan que acceder en detalle a los procedimientos y funciones necesarios para dar soporte a la funcionalidad antes explicada. Del mismo modo, y como ya se ha comentado[1], se han encapsulado en un segundo fichero de extensión de aplicación todos aquellos mensajes de voz imprescindibles para el funcionamiento del sistema de acceso remoto. Gracias a este sistema toda la funcionalidad requerida para el cumplimiento de los requisitos funcionales que nos hemos marcado queda reducida a dos ficheros fácilmente manejables y exportables, de modo que no se requieren complejos sistemas de instalación y mantenimiento. Además gracias a este sistema garantizamos que nuestro interfaz sea independiente de la aplicación base, puediendose emplearse no solo en la aplicación de pruebas que acompaña al presente proyecto fin de carrera, sino desde toda aplicación que requiera de la funcionalidad explicada.

Siguiendo con la filosofía establecida, encapsular la funcionalidad de cada uno de los componentes en módulos independientes, el interfaz se ha desarrollado como un conjunto de clases y objetos que engloban partes funcionalmente distinguibles dentro del conjunto de la funcionalidad del interfaz. Así se proporcionan clases para la reproducción de cada uno de los menús, clases de control de las notificaciones de avisos o clases para el control de los temporizadores, tanto de inactividad de la línea como de notificación de avisos registrados, por ejemplo. De este modo se garantiza que el mantenimiento del interfaz sea mas sencillo y fácil de desarrollar, centrándose dicho proceso en una de estas clases sin necesidad de alterar significativamente el resto de las que forman el interfaz. Esta misma filosofía nos ha dado la posibilidad de reducir todo el sistema de comunicaciones entre el interfaz y la aplicación base a una única clase que proporcionará a la aplicación base todos los procedimientos y funciones necesarios para que ésta pueda acceder a toda la funcionalidad ofrecida por el interfaz. Esto

[1] Véase el apartado 4 Inconvenientes de la solución adoptada

nos refuerza aun más ese carácter de independencia de la aplicación base con respecto al interfaz.

Por último y antes de comenzar con una descripción más detallada de las clases antes mencionadas, cabe destacar que toda la funcionalidad se basa en un conjunto de funciones ofrecidas a su vez por una librería que acompaña al sistema operativo, como ya hemos dicho tipo Windows. Dicha librería, llamada TAPI, se distribuye libremente junto con el sistema operativo por lo que no se requieren de elementos adicionales ni de complejas instalaciones para dar soporte a la funcionalidad ofrecida por el interfaz. Esta librería posee varias versiones, que van desde la 1.3 del antiguo Windows 3.1 hasta la 3.0 de los más modernos sistemas operativos Windows. En nuestro caso nos hemos basado en la versión 2.0 ya que esta versión ofrece soporte a los sistemas operativos de 32 bits sin entrar en la complejidad de la versión 3.0.

Se trata pues de una API(Application Programming Interface) que ofrece todo un conjunto de funciones para el acceso y control de los distintos dispositivos telefónicos instalados en el sistema operativo; así como el acceso y control de las llamadas que por ellos se reciban y se realicen. Así permite controlar y acceder al estado de estos dispositivos, de las llamadas, controlar el proceso de realización y recepción de llamadas, no solo en sistemas simples, hablando desde el punto de vista del número de líneas controladas, como el nuestro, sino también en complejos sistemas como los modernos Call Centers. Su funcionalidad es realmente extensa y muy compleja, ya que abarca todo tipo de funcionalidad ofrecida por los distintos proveedores de servicios telefónicos, TSPs, llamadas en espera, redireccionamiento de llamadas, etc... En el presente proyecto fin de carrera hemos hecho uso de una pequeña parte de todo el potencial de esta API, ya que la gran mayoría de la funcionalidad que ofrece escapa a las intenciones de este proyecto fin de carrera. Para más información sobre TAPI, sus funciones y sus distintas versiones pueden consultarse la documentación mencionada a este respecto en la bibliografía que acompaña a la presente memoria.

Las clases descritas pueden clasificase en grupos funcionalmente distinguibles por lo que nos basaremos en este criterio para realizar una descripción más profunda de cada una de ellas y de sus correspondientes métodos y funciones, aunque se recomienda acudir al Anexo 1 del presente proyecto fin de carrera para acceder a una descripción exhaustiva del código fuente de cada una de las clases.

6.1 Clases de control del interfaz

Las clases clasificadas dentro de este grupo forman el núcleo fundamental de acceso y control a la funcionalidad del interfaz, e implementan los mecanismos de comunicación con la aplicación base. Son por tanto accesibles desde la aplicación base, pudiendo esta crear e instanciar objetos de estas clases. Encapsulan la funcionalidad del resto de las clases que forman el interfaz enmascarándolas y ocultándolas a la aplicación base.

6.1.1 CControlMenu

Se trata de la clase principal del interfaz de acceso remoto. La aplicación base que quiera hacer uso de la funcionalidad ofrecida por el interfaz deberá de instanciar un objeto de esta clase para poder acceder a través de sus métodos a la funcionalidad del interfaz. Por lo tanto sus funciones ofrecen todas las opciones de configuración antes mencionadas[1], funciones para el control y monitorización de la línea telefónica a la que se conecte el interfaz y para la comunicación de los avisos y datos al mismo. Por lo tanto sus funciones pueden agruparse por grupos especializados, que harán uso de otras clases, que ya se irán detallando, para la realización de sus tareas.

Las funciones destinadas al control y monitorización del interfaz proporcionan métodos para la inicialización del interfaz así como para su finalización o el chequeo del estado del mismo.

[1] Véase el apartado 5.1 Configuración del interfaz

Para la inicialización y finalización de la línea telefónica configurada la clase ofrece otro juego de funciones equivalente al anterior, proporcionando métodos para la incialización, finalización y chequeo de la correspondiente línea programada.

Como ya se ha explicado el funcionamiento de del interfaz se basa en un conjunto de parámetros configurables desde el exterior, así que esta clase proporciona las funciones necesarias para el acceso a dichos parámetros. Por lo tanto ofrece todo un juego de funciones para establecer y obtener los valores de todos y cada uno de los parámetros de configuración del interfaz, clave de acceso, número de tonos, línea telefónica, etc...

Finalmente implementa un conjunto de funciones necesarias para el intercambio de datos entre el interfaz y la aplicación base. Por lo tanto ofrece funciones para la comunicación de datos, tales como las emergencias, al interfaz desde la aplicación base. Por otro lado el mecanismo de comunicación de eventos desde el interfaz a la aplicación base se basa en una clase, CMenuSink, también accesible desde la aplicación base, que proporciona las funciones que llamará el interfaz para notificar eventos, tales como peticiones de actuación o de información. Es por ello que la clase también ofrece funciones para el registro de la aplicación base como receptora de los eventos del interfaz con vistas al envío de notificaciones a la misma.

Teniendo en cuenta que, como ya se ha mencionado, la clase hace uso del soporte que le ofrecen las restantes clases que forman el interfaz, el proceso de inicialización del interfaz podría resumirse mediante la siguiente ilustración:

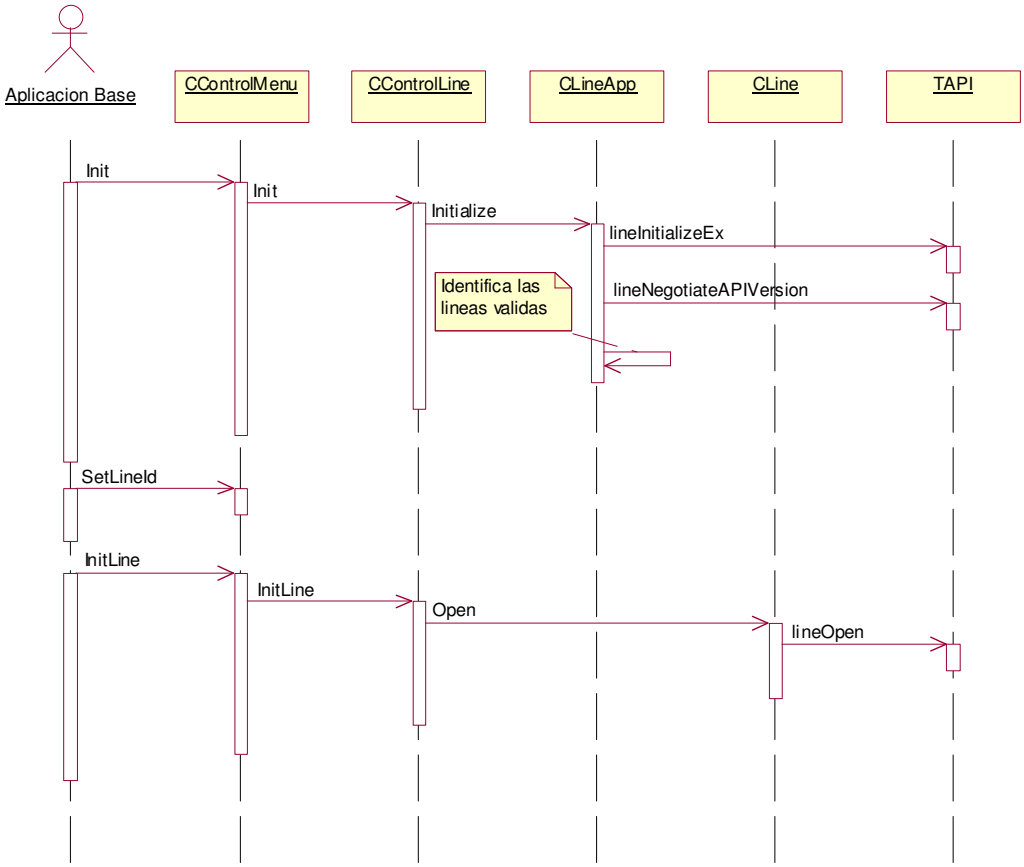

Ilustración 6-1

Por otro lado, la finalización de interfaz estaría organizado tal y como se puede apreciar en el siguiente diagrama.

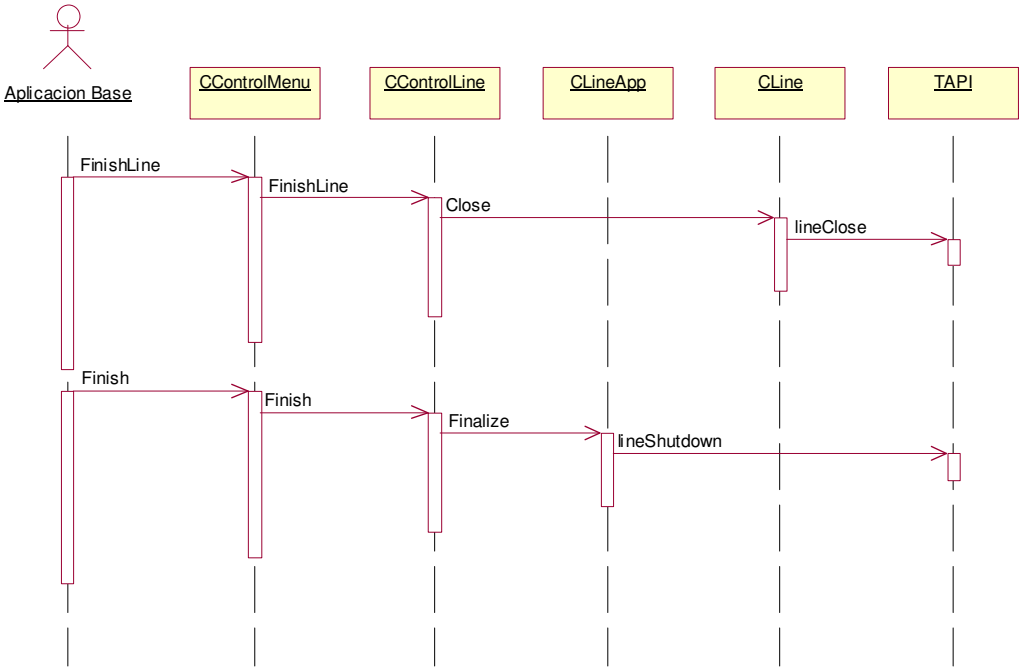

Ilustración 6-2

Por último cabe mencionar que la clase ofrece un conjunto mas amplio de funciones no disponibles para la aplicación base, necesarias para la comunicación entre las restantes clases del interfaz.

6.1.2 CMenuSink

Como ya se ha comentado esta clase ofrece el soporte necesario para la comunicación y envío de datos desde el interfaz hacia la aplicación base. Implementa un conjunto de funciones que la clase principal del interfaz llamará cuando se produzcan los correspondientes eventos. Así la aplicación base instanciará un objeto de esta clase y lo registrará en el interfaz, empleando para ello la correspondiente función de la clase principal del interfaz, como ya se ha señalado. De este modo el interfaz ejecutará la correspondiente función de la clase CMenuSink de la aplicación base cuando se produzca un evento, comunicándoselo de esta forma a la aplicación base.

Los eventos que el interfaz notifica a la aplicación base son tres: Recepción de una nueva llamada, petición de actuación sobre un elemento del sistema y petición de información sobre un elemento del sistema. Por lo tanto esta clase implementa una función para cada uno de estos eventos:

- **OnAction:** Para la notificación de peticiones de actuación.
- **OnInformation:** Para la notificación de peticiones de información.
- **OnNewCall:** Para la notificación de recepción de una nueva llamada.

Cabe destacar el hecho de que las funciones que implementa la clase no realizan función alguna, además de estar declaradas como virtuales, lo que permite que la aplicación base instancie mediante herencia[1] el objeto de esta clase que requiere. Esto permite que la correspondiente clase de la aplicación base implemente la funcionalidad que le sea más adecuada para el tratamiento de los eventos del interfaz sin que por ello se vea alterado o modificado el mecanismo de comunicaciones entre ambos módulos.

6.2 Clases de control de los recursos telefónicos

Dentro de este grupo se engloban todas aquellas clases que ofrecen soporte para el acceso a alguno de los recursos asociados a la línea telefónica. Para ello hacen uso de las funciones y métodos ofrecidos por TAPI, ofreciendo a las restantes clases del interfaz una capa de comunicación y acceso a los recursos de TAPI. No están accesibles a la aplicación base y su ámbito se restringe al interior del módulo del interfaz, ya que su función es la de servir como soporte para las restantes clases del interfaz. Por lo tanto la aplicación base accede a los recursos de estas clases a través de la clases mencionadas en el apartado anterior.

6.2.1 CControlLine

Forma el núcleo de control a los recursos telefónicos, centralizando las funciones de inicialización y finalización de los elementos centrales requeridos para el funcionamiento del interfaz. Así es la encargada de inicializar el interfaz, inicializando TAPI, y de inicializar la línea

[1] Mediante la herencia una clase hace suyas las variables y funciones de la clase de la que hereda

telefónica, así como de proporcionar métodos para el acceso a los datos principales de la configuración y estado de las conexiones. Es instaciada por la clase principal de control del interfaz y usada por esta para la inicialización y finalización de los recursos telefónicos, para lo cual hace uso de otras clases de este grupo que se comentarán mas adelante. Por lo tanto ofrece funciones para la inicialización y finalización de TAPI, así como para la inicialización y finalización de la línea seleccionada.

Como soporte adicional implementa un conjunto de funciones destinadas al acceso a los datos de los objetos empleados para dar soporte a las funcionalidades anteriormente mencionadas, y que son de interés para las restantes clases del interfaz.

Finalmente como clase principal de control de la línea telefónica es la encargada de determinar y controlar el mecanismo de recepción de los eventos ocurridos en la línea. El funcionamiento de TAPI se basa en la generación de eventos ante los distintos hechos que ocurren en la línea telefónica. Estos deben ser recogidos y procesados por el módulo que hace uso de TAPI para poder actuar en consecuencia al evento producido. Así la recepción o finalización de una llamada genera un determinado evento[1] que al ser recepcionado permite saber si se esta recibiendo una llamada nueva o por si el contrario se está finalizando. Para dar soporte a este mecanismo de comunicación, se ha desarrollado una clase, CLineSink similar a la ya mencionada CMenuSink, que al igual que este última implementa un conjunto de funciones que son llamadas cuando se produce un determinado evento en la línea. Es responsabilidad de CcontrolLine el registrar la correspondiente instancia de ClineSinl que recepcionará los eventos.

6.2.2 CLineSink

Esta clase es el soporte básico para la comunicación de eventos ocurridos en la línea al interfaz de acceso remoto. Implementa un conjunto de funciones que son llamadas cuando se produce un determinado evento en la línea.

[1]Véase el Anexo 1 TAPI.

El sistema empleado por TAPI para la notificación de eventos se basa en la ejecución de una función[1] establecida al abrir la línea. Esta función es ejecutada automáticamente cada vez que se produce un evento, y uno de los parámetros de esta función es un valor igualmente establecido al abrir la línea. De esta forma estableciendo dicho parámetro como la referencia a un objeto de la clase CLineSink podremos llamar a la correspondiente función al recibir un evento. Así las funciones que implementa esta clase son:

- **OnReply:** Llamada cuando se recibe un evento LINE_REPLY como respuesta a la finalización de una petición de actuación sobre la línea.
- **OnCallState:** Llamada como respuesta al evento LINE_CALLSTATE producido cuando se produce un cambio en el estado de una llamada.
- **OnDevState:** Llamada cuando se recibe un evento LINE_LINEDEVSTATE producido por cambios en el estado de la línea.
- **OnAppNewCall:** Ejecutada como respuesta al evento LINE_APPNEWCALL producido ante la llegada de una nueva llamada.
- **OnMonitorDigits:** Llamada como respuesta a un evento LINE:MONITORDIGITS que se recibe cuando se recibe la pulsación de una tecla.
- **OnGatherDigits:** Ejecutada cuando se recibe un evento LINE_GATHERDIGITS producido cuando se recibe una secuencia de pulsaciones de teclas.
- **OnLineClose:** Llamada como respuesta al evento LINE_CLOSE producido cuando se cierra la línea.

En nuestro sistema el recepcionador de eventos de la línea es siempre la clase principal del interfaz, ya que es la encargada de hacer de puente entre las distintas clases que forman el interfaz. Así se registra como receptara de eventos y redirige los mismos hacia las demás clases según el estado del interfaz y de la llamada en curso, en caso de haber alguna.

6.2.3 ClineApp

[1] A este tipo de funciones se les denomina funciones de CALLBACK

Esta clase es la primera empleada por CControlLine para acceder a los recursos telefónicos. Ofrece un interfaz básico de inicialización y finalización de TAPI, así como funciones para una gestión básica de las líneas detectadas por TAPI en el sistema. Des este modo las llamadas a TAPI quedan enmascaradas por el interfaz de esta clase, pudiéndose modificar los parámetros de incialización de la librería sin que se vea alterado el proceso necesario para el arranque de nuestro interfaz telefónico.

Es la encargada de llamar a la función de inicialización de TAPI[1], y establecer los parámetros necesarios para que este ofrezca toda su funcionalidad a nuestro interfaz telefónico. Entre otros parámetros que establece esta la función de recepción de eventos de TAPI comentada anteriormente, llamada *lineCallbackFunc*. Como ya se ha comentado esta función será la responsable de recepcionar los eventos que TAPI genere como respuesta a los cambios producidos en la línea, y reenviarlos al objeto CLineSink indicado.

Del mismo modo CCntrolLine recurre a esta clase para la finalización de TAPI mediante la llamada a la función *Finalize*, la cual accede a la correspondiente función de finalización de TAPI[2].

Por último ofrece un conjunto de funciones de acceso a los parámetros de arranque y de soporte de las líneas, con vistas a que las clases superiores puedan acceder a estos datos en caso de requerirlos para algún tipo de operación, como por ejemplo la apertura de una línea.

6.2.4 CLine

Esta clase ofrece un interfaz básico de acceso a las líneas telefónicas. Es empleada por CControlLine para abrir y cerrar la línea telefónica que será usada para dar soporte al interfaz de acceso telefónico, además de emplearla como herramienta de gestión de la línea.

[1] lineInitializeEx, para más detalles consúltese el Anexo 1: TAPI
[2] lineShutDown, para más detalles consúltese el Anexo 1:TAPI

Dada su funcionalidad la clase proporciona dos métodos básicos, uno para la apertura de la línea, *Open*, y otro para su cierre, *Close*. Su funcionamiento se soporta en sendas llamadas a las funciones que TAPI proporciona para abrir y cerrar una línea telefónica[1], enmascarando de este modo la librería a las clases superiores que hagan uso de sus recursos.

Como responsable de la apertura y mantenimiento de la línea, determina parámetros básicos en el funcionamiento del interfaz, tales como el objeto CLineSink que recibirá las notificaciones de eventos tal y como ya se ha comentado, operación que realiza al abrir la línea. Además proporciona acceso a datos de gestión de la línea abierta, permitiendo a las clases superiores el acceso a los mismo para su uso en otras funciones.

6.2.5 CCall

Es la clase encargada de implementar el acceso a los métodos necesarios para la gestión de una llamada en particular. Así es empleada por las restantes clases tanto para el acceso a una llamada entrante como para la realización de una llamada saliente, funcionado como controlador de la llamada a la que se asocia.

Dada su funcionalidad el interfaz implementado por la clase proporciona todo un conjunto de métodos destinados a la gestión y configuración de una llamada, permitiendo a las clases superiores realizar operaciones tales como descolgar, *Answer*, colgar, *Drop*, o realizar una llamada, *MakeCall*. También unifica en una sola clase los procesos de recepción y realización de llamadas, ya que el manejador de la llamada, variable del tipo HCALL que emplea TAPI para identificar la llamada[2], puede ser establecido desde el exterior de la clase o desde dentro de la misma, al realizar una llamada mediante el método *MakeCall*.

Al igual que las clases anteriores, CLineApp y CLine, CCall hace un uso exhaustivo de las funciones proporcionadas por TAPI en lo referente a la gestión de las llamadas. Cada uno de sus métodos se basa en la correspondiente función dada por TAPI, por lo que gracias a ello las

[1] lineOpen y lineClose respectivamente, consúltese el Anexo 1: TAPI para un mayor detalle
[2] TAPI emplea manejadores únicos para identificar las llamadas, las líneas, HLINE, y las conexiones a TAPI, HLINEAPP

clases superiores acceden a la funcionalidad que requieren para el desarrollo de sus correspondientes operaciones sin tener que recurrir directamente al soporte dado por la librería. Así por ejemplo las funciones *Answer* y *MakeCall* emplean las correspondientes funciones lineAnswer y lineMakeCall de TAPI[1].

6.3 Clases de datos de los recursos telefónicos

Dentro de esta clasificación se agrupan todas aquellas clases que aún basándose en el soporte dado por TAPI no ofrecen funciones de gestión y/o configuración de alguno de los recursos telefónicos accesibles desde la librería. Por el contrario ofrecen acceso a datos sobre dichos recursos, tales como configuraciones, estado o capacidades. Dada su naturaleza las clases recogidas bajo esta clasificación también están disponibles para la aplicación base, ya que su funcionalidad va mas allá de la meramente funcional, pudiendo proporcionar datos útiles a la aplicación base para alguna de sus posibles funciones, tales como listados de las líneas detectadas, los países configurados, etc...

En su conjunto se basan en la realización de una llamada a la correspondiente función de TAPI y la obtención a partir de ella de los datos asociados al elemento accedido. Este tipo de datos se recogen bajo una estructura, distinta según la función de TAPI empleada, proporcionando cada una de estas clases los métodos necesarios para acceder a cada una de las variables que forman su estructura de datos. Las distintas estructuras empleadas por TAPI tienen la particularidad de tener una longitud variable[2], de forma que está puede cambiar de un acceso a otro, por lo que todas las clases deben ajustar el tamaño de sus respectivas estructuras al acceder a TAPI en busca de los datos solicitados.

6.3.1 CAddressCaps

Esta clase accede a los datos sobre las capacidades telefónicas de una determinada dirección especificada sobre una línea. Como ya se ha comentado su funcionamiento se basa

[1] Consúltese el Anexo 1: TAPI para más información
[2] Se trata de estructuras de longitud variable, VLS, que requieren de un procesado especial para su actualización con datos válidos, véase el Anexo 1: TAPI

en una función de TAPI, concretamente lineGetAdressCaps, que rellena la estructura de datos, del tipo LINEADDRESSCAPS, con la información que ha recuperado de la dirección. Por lo tanto, y como el resto de las clases de esta clasificación, su empleo por parte de otras clases o procedimientos se hace en dos pasos.

- En un primer paso se llama a la función que hace la llamada a la función de TAPI que accederá a los datos, tras establecer los parámetros adecuados.
- En un segundo paso se accede a los datos deseados mediante las correspondientes funciones habilitadas por la clase

En nuestro caso es el método *GetAddressCaps* el que accede a los datos de la dirección. Los parámetros que necesita para acceder a los datos son tres: una referencia al controlador de inicialización de TAPI, un objeto de la clase CLineApp, establecido mediante el método *SetLineApp*, el identificador de la línea y el identificador de la dirección. Con estos tres parámetros el método actualiza la estructura de datos con los de la dirección dada.

Los métodos de acceso a los datos se corresponden a cado uno de los campos de la estructura de datos. Así la clase ofrece entre otros, la estructura LINEADDRESSCAPS es muy amplia y posee numerosos campos[1], métodos para acceder al estado de la dirección, *GetAddressStates*, o para acceder a los estados comunes a todas las llamadas que se realicen a través de la dirección, *GetCallInfoStates*.

6.3.2 CAddressStatus

Esta clase accede a los datos sobre el estado actual de una dirección especificada sobre una línea. En este caso la función de TAPI empleada para ello es lineGetAddressStatus y la estructura de datos que recoge el estado de la dirección es del tipo LINEADDRESSSTATUS.

Para acceder a los datos de la dirección se llamaría al método *GetAddressStatus* indicándole la dirección sobre la que se desea obtener la información mediante el identificador de la misma y la línea sobre la que se establece la dirección mediante el manejador de dicha

[1] Consúltese la bibliografía sobre TAPI para más detalles sobre la estructura LINEADDRESSCAPS

línea. Posteriormente se haría uso de los métodos de acceso a los datos para acceder a los distintos campos de la estructura, tales como *GetNumActiveCalls* para obtener el número de llamadas activas sobre la dirección o *GetAddressFeatures* que determina las acciones que se pueden realizar sobre las llamadas establecidas sobre la dirección.

6.3.3 CCallInfo

Como su nombre indica esta clase accede a la información genérica disponible sobre una llamada determinada. Así la llamada a TAPI que emplea para acceder a los datos es lineGetCallInfo y la estructura de datos que recoge la información sobre la llamada es del tipo LINECALLINFO.

Método de inicialización de la estructura de datos, *GetCallInfo*, requerirá solo del manejador que identifica a la llamada para poder acceder a los datos de la misma. Por otro lado, y al igual que en la clase CAddressCaps, la estructura de datos que maneja esta clase es muy amplia y posee numerosos campos referidos a multitud de parámetros de información de la llamada. Por lo tanto se trata de una clase muy amplia con numerosos métodos, se recomienda consultar la bibliografía sobre TAPI indicada para obtener más información. Entre todos ellos destacan los fétodos de acceso a los modos de datos de información[1], tanto soportados por la llamada como el actualmente establecido; de acceso a los modos de teclas disponibles, determinan si la llamada detecta pulsos o tonos como indentificadores de los números; y de acceso a los números de teléfono llamante y llamado.

6.3.4 CCallStatus

Esta clase accede a los datos sobre el estado de la llamada indicada, así como otros datos que dependen de las capacidades de la dirección de la línea sobre la que se ha establecido. Para acceder a estos datos emplea la función lineGetCallStatus de TAPI que recoge la información obtenida en una estructura del tipo LINECALLSTATUS.

[1] Media Modes en inglés, estable si la llamada es un fax, una conexión de datos o de voz.

Tras llamar al método *GetCallStatus*, encargado de completar la estructura de datos, se llamarán a los distintos métodos habilitados por la clase para acceder a los distintos campo que forman la estructura LINECALLSTATUS. Entre ellos caben destacar *GetCallStateMode*, que nos determina el estado actual de la llamada, y *GetCallPrivilege*, que nos da los privilegios que se tienen sobre la llamada, estos privilegios determinan el tipo de eventos que se recibirán de la llamada.

6.3.5 CCountryList

Esta clase es la encargada de listar los países disponibles así como de acceder a los datos relacionados con cada uno de ellos, incluidas sus correspondientes reglas para la realización de llamadas, tales como los identificadores telefónicos. Para ello recurre a la función lineGetCountry de TAPI y a la estructura de datos LINECOUNTRYLIST.

El método de inicialización de la estructura de datos, *GetCountries* solo requiere de la versión de TAPI que se este empleando, dato disponible desde la clase de control del interfaz. Tras realizar la llamada a dicho método se accedería a los datos de cada país mediante los restantes métodos facilitados por la clase, tales como el número de países obtenidos, *GetNumCountries*, el nombre del mismo, *GetCountryName*, o el código del país, *GetCountryCode*.

6.3.6 CDeviceID

Esta clase es la encargada de obtener el identificador del dispositivo asociado a una llamada, dirección o línea. Dicho identificador será el elemento básico para la transmisión y recepción de voz a través de la línea. Se basa en la función lineGetID de TAPI.

Su funcionamiento se basa en tres funciones, cada una especializada en el acceso a un determinado tipo de dispositivo. Así tenemos un método para el acceso a una línea, *GetIDFromLine*, a una dirección, *GetIDFromLineAddress*, o a una llamada, *GetIDFromCall*. Estas tres funciones requieren, todas, ellas de una cadena de caracteres que identifique el

dispositivo del que se desea obtener el identificador. El valor de dicha cadena de caracteres puede ser uno de los listados en la siguiente tabla.

comm	Puerto serie de comunicaciones
comm/datamodem	MODEM a través de un puerto serie de comunicaciones
comm/datamodem/portname	Nombre del puerto al que está conectado el MODEM
wave/in	Dispositivo de sonido solo de entrada
wave/out	Dispositivo de sonido solo de salida
midi/in	Secuenciador MIDI solo de salida
midi/out	Secuenciador MIDI solo de entrada
ndis	Dispositivo de red
tapi/line	Dispositivo de linea
tapi/phone	Dispositivo de microfono
tapi/terminal	Dispositivo de terminal

6.3.7 CLineDevCaps

Esta clase accede a los datos sobre las capacidades telefónicas de una determinada línea, información que es válida para todas las direcciones de la establecidas sobre la misma. Para ello la clase hace uso de la función lineGetDevCaps de TAPI que completa una estructura del tipo LINEDEVCAPS con los datos obtenidos de la línea. Esta estructura posee un elevado número de componentes, siendo muy extensa la información obtenida sobre la línea.

Para hacer uso de la funcionalidad proporcionada por la clase se deben seguir tres pasos. Primeramente se establece, mediante el método *SetLineApp*, la referencia al objeto ClineApp empleado para inicializar TAPI. Seguidamente se actualiza la estructura de datos mediante la llamada al método *GetDevCaps* indicándole el identificador de la línea sobre la que se desea obtener la información. Por último de llaman a los métodos asociados a los campos de información a los que se desea acceder, como por ejemplo al método *GetMediaModes* para obtener los modos soportados por la línea.

6.3.8 CTranslateCaps

Esta clase hace uso de la función lineGetTranslateCaps de TAPI y de la estructura LINETRANSLATECAPS. Su función es la de acceder a los datos empleados por TAPI para realizar las conversiones entre los distintos tipos de formatos de numeración telefónica[1]. Su principal interés radica en que a través de esta clase podemos acceder a la lista de localizaciones definidas en el sistema. Estas localizaciones se asocian a distintas configuraciones programadas en el sistema operativo del ordenador y que determinan como se transformarán los números telefónicos para su marcado.

Su funcionamiento se basa en el método *GetTranslateCaps* llamado tras establecerse la referencia al objeto de la clase CLineApp empleado para inicializar TAPI. Tras ello se podrá acceder a la lista de localizaciones, así como a los restantes datos almacenados por la estructura LINETRANSLATECAPS, como por ejemplo a los nombres de las localizaciones mediante el método *GetLocationName*.

6.3.9 CTranslateOutput

Como ya se ha comentado existen varios formatos para representar un mismo número telefónico, pudiéndose convertirse entre ellos. Es esta clase y los métodos que ofrece los encargados de realizar estas conversiones. Para ello emplean la función de TAPI lineTranslateAddress y la estructura LINETRANSLATEOUTPUT.

Su funcionamiento, al igual que las demás clases de esta clasificación, se basa en una llamada a un método que redirige la petición de datos hacia TAPI. En nuestro caso se trata del método *TranslateAddress* al cual se le indica la línea sobre la que se trabaja y el número telefónico, en cualquiera de los formatos. Tras esta llamada se podrá acceder al correspondiente formato asociado el número dado empleando para ello los correspondientes métodos, *GetDialableString* para el formato marcable y *GetDisplayableString* para el mostrable.

[1] Existen dos tipos de formatos de numeración: el mostrable, que todos usamos para apuntar un número de teléfono, y el marcable, que además emplea un juego de caracteres especiales de control y que emplea TAPI para marcar. Véase el Anexo 1: TAPI.

6.4 Clases de control y gestión de los menús

Bajo esta denominación podemos agrupar todas aquellas clases relacionadas directamente con los menús que dan soporte a la funcionalidad del interfaz. Así estas clases son las responsables de reproducir los mensajes de voz, así como de controlar las correspondientes pulsaciones de las teclas por parte del usuario. Ofrecen funciones para el control de la reproducción del menú así como para la notificación de los eventos de pulsación de las teclas. Con el fin de unificar los accesos a estas clases desde el objeto de la clase de control del menú, y puesto que es este el encargado de recibir los eventos de la línea y redirigirlos a los correspondientes objetos, todas las clases de este conjunto heredan de una clase virtual única que implementa los métodos comunes a todas ellas. De este modo aseguramos un acceso unificado a todas ellas y unos métodos y un procedimiento de empleo común.

Como ya se ha mencionado su funcionalidad gira en torno a un mensaje de voz pregrabado que constituye en mensaje que se reproducirá a través de la línea telefónica y que constituye la base del menú. Para ello cada una de las clases de este conjunto mantiene un objeto de la clase CWave[1] que le proporcionará el acceso a los recursos necesarios para reproducir dicho mensaje. Cabe destacar que el menú pregrabado se haya localizado, como ya se ha explicado, en un fichero de extensión desarrollado a tal fin y distinto al que contiene el interfaz en sí.

6.4.1 COpcionMenu

Como ya se ha hecho mención las clases de esta clasificación heredan de una clase única que implementa los métodos básicos y comunes para que desarrollen su funcionalidad, pero que no podrá darles una funcionalidad definida ya que esta depende del menú en sí mismo. Se trata pues de una clase virtual, no ofrece funcionalidad aunque si define un conjunto de métodos y sus prototipos, conformando así un interfaz común para todas las clases reales que hereden de ella. Es la clase COpcionMenu la encargada de desarrollar este soporte en nuestro caso.

[1] Véase el apartado 6.8.1 CWave para más detalles

Teniendo en cuenta que deberá implementar solo aquellos métodos comunes a todas las clases de la clasificación, los métodos a los que da soporte son:

- **Open:** Destinado a abrir el menú, es decir, es la función que se empleará para iniciar el menú y reproducir el mensaje de voz asociado, en caso de ser único, o de iniciar la reproducción de los mensajes de voz, en caso de ser varios.
- **Close:** Su finalidad es la de cerrar el menú, liberando los recursos ocupados durante su actividad.
- **OnDigit:** Empleada para notificar a las clase los eventos de pulsación de una tecla. Es empleada por el objeto de la clase principal del interfaz para notificarle al menú dichos eventos. A tal efecto los distintos menús deberán registrase en la clase principal del interfaz durante su proceso de apertura, proceso facilitado por echo de que todos ellos deriven de esta clase virtual.
- **OnGatherDigits:** Al igual que la anterior es empleada para notificar a la clase del menú eventos ocurridos en la línea, aunque en este caso el evento notificado es el de la recepción de una secuencia de pulsaciones de teclas.
- **OnWaveOutDone:** En este caso se trata de el método empleado para la notificación a la clase de la finalización de la reproducción del mensaje de voz reproducido. Como ya se verá más adelante[1] el mecanismo empleado para reproducir los mensajes de voz se basa, al igual que TAPI, en una función que se ejecuta cada vez que se produce un evento, en este caso asociados a la reproducción de los mensajes de voz, desde la cual se redirigirá el evento.

6.4.2 CMenuInicio

Implementa el **Menú de Inicio**[2] del interfaz. Su funcionamiento se basa en un único fichero de voz, identificado como IDR_WAVE_MENU_INICIO dentro del fichero de mensajes, reproducido en la correspondiente llamada al método *Open* de la clase. Como ya se ha comentado en dicho método la clase se registra como menú activo en la clase de control del

[1] Apartado 6.7 Clases de reproducción de voz.
[2] Véase el apartado 5.2.1 Menú de Inicio para una funcionalidad más detallada de este menú.

interfaz con el propósito de establecerse como destinatario de los eventos que se produzcan en la línea, empleando para ello en método *SetMenuOption* de dicha clase.

Por otro lado el método *OnWaveOutDone* de la clase se limita a habilitar la recepción de las pulsaciones de las teclas, sin realizar ningún otro tipo de tratamiento, ya que, como se ha dicho, este menú funciona en base a un único mensaje de voz.

Finalmente el método *OnDigit*, responsable de la recepción de los eventos de pulsación de las teclas, controla la tecla pulsada, pasando a abrir el siguiente menú, a terminar la comunicación o seguir esperando una pulsación según la tecla recibida.

6.4.3 CMenuAcceso

En este caso la clase implementa el **Menú de Acceso**[1] que al igual que el menú anterior se basa en un único fichero de voz, identificado como IDR_WAVE_MENU_ACCESO en la librería de mensajes.

Nuevamente la función *OnWaveOutDone* tampoco realiza un tratamiento especial tras la finalización de la reproducción del mensaje de voz del menú.

Sin embargo el método *OnDigit* si desarrolla una funcionalidad más compleja que en la clase anteriormente comentada. Esta se debe a que la clase implementa la funcionalidad de recepción y control de la clave de acceso al sistema. De este modo cada una de las pulsaciones de teclas que se produzcan serán controladas conforme a las ya recibidas, comprobándose que se recibe correctamente la secuencia de cinco números seguidos de la tecla '#'. Si se obtiene una secuencia válida, el método compara la secuencia obtenida con la programada en la clase de control del interfaz para proporcionar acceso al siguiente menú.

[1] Véase el apartado 5.2.2 Menú de Acceso para más detalles de su funcionalidad.

6.4.4 CMenuOpciones

Esta clase es la responsable de implementar el **Menú de Opciones**[1]. Dada la simplicidad del mismo, los métodos de la clase no realizan funciones especializadas, salvo abrir el correspondiente mensaje de voz, IDR_WAVE_MENU_OPCIONES, y controlar la tecla pulsada para actuar en consecuencia. Cabe destacar el echo de que, como se verá en el apartado siguiente, se registra como antecesor en las dos siguientes clases.

6.4.5 CMenuAccion

Implementa la funcionalidad asociada al **Menú de Acción**[2] por lo que sus métodos desarrollan una funcionalidad más especializada, ofreciendo además otros propios de su funcionalidad.

Así el método *Open*, en el que se abre el correspondiente mensaje de voz, IDR_WAVE_MENU_ACCION, da paso a una secuencia de pulsaciones encaminadas a establecer el sistema sobre el que se desea actuar y el valor de actuación. De este modo el método de recepción de las pulsaciones de las teclas, *OnDigit*, se especializa para controlar la correcta recepción del código de tres cifras que identifica el sistema y del valor de actuación.

La funcionalidad del menú establece dos métodos propios para la clase encaminados a reproducir mensajes de error, *Error* que reproduce el mensaje IDR_WAVE_ERROR, y mensajes con el resultado de la acción sobre el sistema, *Success* y los mensajes IDR_WAVE_SUCCESS, cuando la actuación se completa satisfactoriamente, e IDR_WAVE_NO_SUCCESS, cuando no se puede completar la actuación.

Finalmente el método *OnWaveOutDone* controla la finalización de la secuencia de actuación sobre el sistema. Habilita la recepción de eventos de pulsación de teclas en caso de que dicha secuencia no se haya finalizado o pasa al menú registrado como predecesor, mediante el método *SetMenuPrevio*, en caso de haberse finalizado dicha secuencia.

[1] Véase el apartado 5.2.3 Menú de Opciones para una descripción más detallada del mismo.
[2] Véase el apartado 5.2.4 Menú de Acción.

6.4.6 CMenuInformacion

Da soporte a la funcionalidad del **Menú de Información**[1], por lo que dada la funcionalidad de dicho menú, la clase desarrolla unos procedimientos más complejos e implementa un conjunto de métodos propios destinados a completar su funcionalidad.

Al igual que en la clase anterior en método *Open*, que reproduce el mensaje de voz principal del menú, IDR_WAVE_MENU_INFORMACION, da paso a la secuencia de pulsaciones de teclas para la introducción del código numérico de tres cifras identificativo del elemento sobre el que se desea pedir información. Por lo tanto en método *OnDigit* se especializa para la recepción y control de dicha secuencia.

La clase se encarga de reproducir el correspondiente mensaje necesario para comunicar al usuario la información solicitada. Es por ello que la clase implementa un conjunto de métodos muy especializados destinados a reproducir las distintas partes que forman dicho mensaje. Así el método *OnWaveOutDone* resulta considerablemente más complejo que en casos anteriores, ya debe realizar un control de la secuencia de reproducción de las distintas partes del mensaje de información. Internamente se hace uso de tres métodos, *PlayStartMessage*, *PlayValue* y *PlayEndMessage*, para reproducir, respectivamente, el mensaje de comienzo, el valor y el mensaje de finalización. El primero de ellos reproduce el mensaje de comienzo que se establezca desde la aplicación base como respuesta a la petición de información que le hace el interfaz, mensaje que por lo tanto resulta configurable para el usuario del sistema. Del mismo modo la función *PlayEndMessage* reproduce el mensaje de finalización que se haya establecido en dicha respuesta. En cuanto al método *PlayValue* analiza la información dada sobre el elemento accedido y sus características para, haciendo uso de un conjunto de mensajes pregrabados, reproducir el estado del elemento solicitado, ya sea numérico o digital, para lo que empleará el método *GetWave*.

[1] Véase el apartado 5.2.5 Menú de Información.

El proceso genérico de funcionamiento de esta clase puede resumiese mediante la siguiente figura.

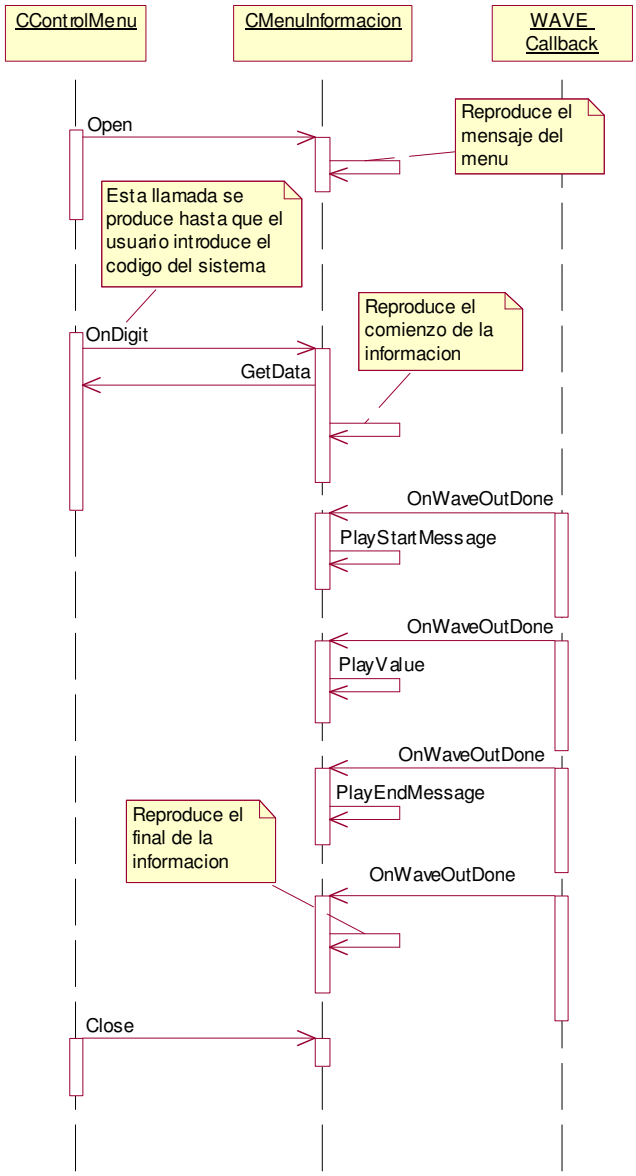

Ilustración 6-3

Finalmente la clase emplea los mismo métodos *Error* y *Success* para informar al usuario sobre los posibles errores al introducir el código numérico e informar sobre el resultado de la operación de petición de información, una vez dada esta en caso de haberse realizado correctamente.

6.5 Clases de control y gestión de los avisos

Dentro de este grupo se encuentran aquellas clases que proporcionan al interfaz el soporte necesario para poder recibir y gestionar los avisos y emergencias que desde la aplicación base se le notifiquen. Por lo tanto su funcionalidad se puede dividir en dos campos diferenciados dentro de su ámbito de actuación:

- Proporcionan los métodos necesarios para que la aplicación base establezca los avisos y emergencias junto con los datos necesarios para su reproducción.
- Implementan la funcionalidad requerida para el almacenaje, reproducción y gestión de los avisos y emergencias comunicados.

La funcionalidad se distribuye entre un objeto principal de control encargado de gestionar la lista de avisos/emergencias que se hayan ido produciendo y de realizar y controlar las correspondientes llamadas, como ya se ha comentado[1] el mecanismo empleado para notificar un evento al usuario se basa en la realización de una llamada al número de teléfono indicado y la reproducción del mensaje de voz indicado. Además de este objeto de control se emplearan un conjunto de objetos para almacenar los avisos que se hayan ido produciendo. De este modo en esta clasificación contamos con dos clases, ninguna de las cuales estará disponible para la aplicación base por tratarse de elementos propios del interfaz y empleados exclusivamente por él.

6.5.1 CControlAverias

Implementa la funcionalidad del objeto de control del sistema de recepción, gestión y comunicación de avisos, por lo que el clase de control del interfaz, CControlMenu, posee un objeto de esta clase. Gracias a esta división podríamos modificar los mecanismos para la gestión interna de los avisos sin que el resto de las funcionalidades del interfaz se vean significativamente alteradas.

[1] Véase el apartado 5.3 Realización de llamadas y envío de avisos.

Dentro del conjunto de métodos que esta clase implementa pueden distinguirse entre dos conjuntos diferenciados: aquellos destinados a configurar los parámetros necesarios para la realización de las llamadas; y otro grupo destinado a la gestión de los avisos.

El primero de ellos implementa las funciones para establecer y acceder a los dos parámetros necesarios para la realización de la llamada, el número de teléfono al que se llamará, *SetCalledNumber* y *GetCalledNumber*, y el código del país del dicho número telefónico, *SetCalledCountryCode* y *GetCalledCountryCode*. Dichos parámetros son configurables desde la aplicación base encargándose la clase CControlMenu de editarlos hacia el exterior, ya que al no estar disponible la clase para la aplicación base sus métodos no son directamente alcanzables desde el exterior del interfaz.

El segundo de los conjuntos implementa los métodos destinados a la gestión de los avisos. Así se implementan funciones para el establecimiento de nuevos avisos, *Add*, para su eliminación, *Remove*, para su notificación, *Notifica*, y para la finalización de la notificación, *Finaliza*.

Hay que destacar el echo de que la clase implementa, heredando de la clase CLineSink, los métodos de recepción de eventos de la línea. Esto es así ya que realiza llamadas para comunicar los mensajes de aviso, por lo que debe de llevar el control de dichas llamadas. Esta funcionalidad implica ser la destinataria de los eventos que se produzcan y por tanto debe de implementar los correspondientes métodos.

La gestión de los avisos se basa en una lista de objetos de la clase CAviso, explicada a continuación, encargados de mantener los datos de los avisos. Dicha lista es mantenida por la clase, almacenando en ella los avisos que se producen de forma ordenada, de mayor a menor, según su prioridad. En esta lista se almacenarán nuevos avisos, mediante el método *Add*, o se eliminarán, método *Remove*. Como dichos métodos no están accesibles desde el exterior del interfaz, la clase principal del interfaz desarrolla un método para exportar el mecanismo de introducción de nuevos avisos hacia el exterior, que internamente realiza una llamada al

método *SetAviso* encargado de generar el objeto CAviso y almacenarlo en la lista. De este modo aislamos el procedimiento de gestión de los avisos de la aplicación base y de la clase principal del interfaz.

Por otro lado el proceso de notificación de avisos es controlado por la clase principal de control del interfaz, CControlMenu, que lo inicia cada cierto tiempo[1] siempre y cuando no haya un proceso de recepción de llamadas en curso. Dicho procedimiento se inicia con una llamada el método *Notifica* encargado de iniciar el proceso, recuperando el aviso de mayor prioridad, eliminando aquellos ya notificados y realizando la llamada al correspondiente número telefónico. En este punto cabe destacar el hecho de que la clase maneja llamadas, salientes en este caso, por lo que deberá ser la receptora de los eventos que estas llamadas produzcan en la línea. Con tal fin el método *Notifica* registra a la clase en el objeto de la clase de control del interfaz como receptara de dichos eventos. Así una vez recibido el evento de establecimiento de la llamada[2], en el método de recepción del estado de la llamada, *OnCallState*, se indicaría al aviso anteriormente recuperado que se reprodujera. A partir de este momento el control del aviso se realiza mediante la función de recepción de pulsaciones de teclas, *OnMonitorDigits* que finalizaría la llamada en caso de detectar una pulsación y marcaría el aviso como notificado para su posterior eliminación. En caso de no recibirse dicha pulsación será el propio aviso el de que sé de por finalizado sin marcase como notificado. En ambos casos se realizaría una llamada al método *Finaliza*, en el primer caso indicándole que el aviso activo ha sido notificado.

[1] Véase el apartado 6.6 Clases de temporización.
[2] Véase el Anexo 1: TAPI para mayor detalle de los eventos de la línea.

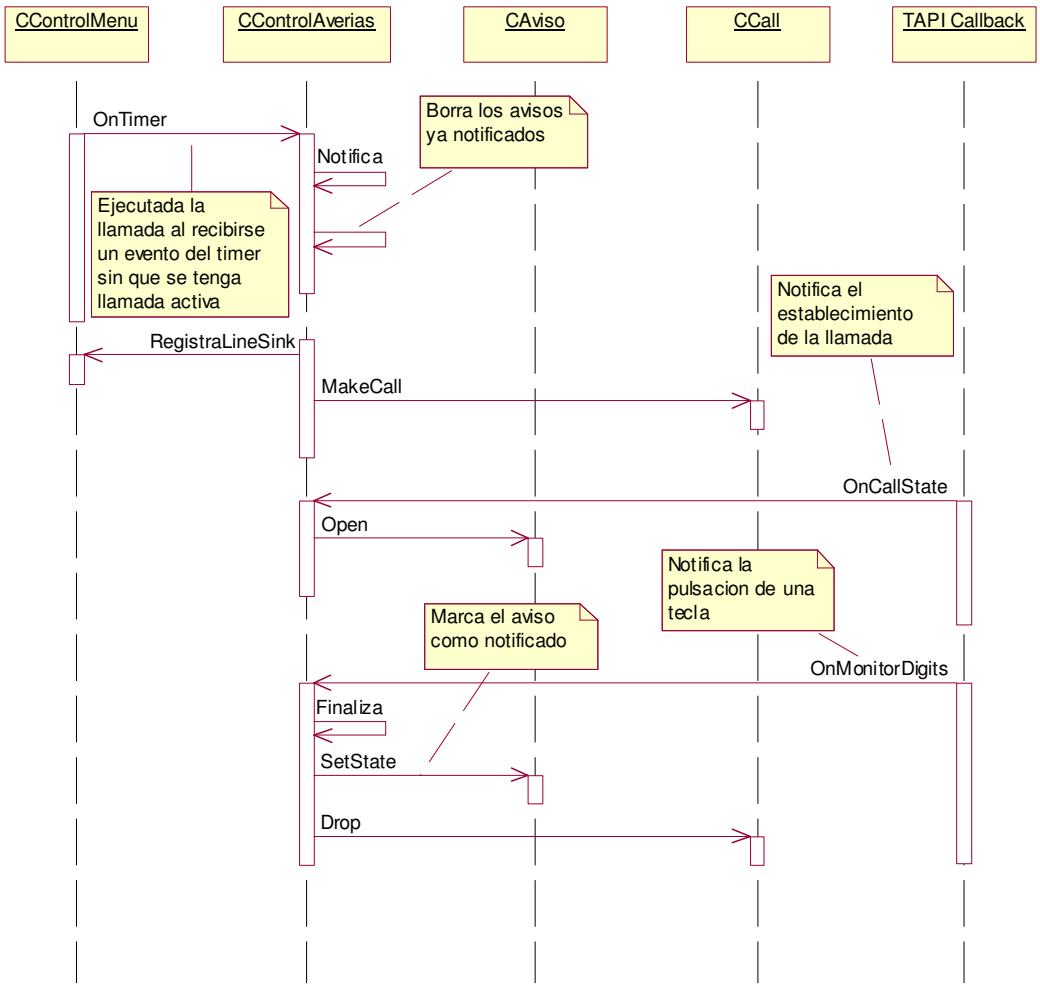

Ilustración 6-4

6.5.2 CAviso

Las funciones básicas de esta clase se distribuyen entre proporcionar un soporte para almacenar los datos relacionados con un determinado aviso, proporcionando los métodos necesarios para el acceso a los mismos. e implementar el mecanismo para la reproducción y control del aviso que representa.

De este modo la clase desarrolla un conjunto de métodos para el establecimiento y acceso los parámetros del aviso, prioridad, mensaje y estado. El primero de ellos, accedido mediante los métodos *SetPriority* y *GetPriority*, se emplea, como ya se ha comentado, para establecer una clasificación entre los mensajes con el fin de proporcionar al usuario un método que

garantice la reproducción de aquellos mensajes más importantes. El segundo de los parámetros, cuyo acceso se realiza a través del método *SetWave*, determina el fichero de voz que será reproducido durante el proceso de notificación del aviso. Por último, el estado representa el estado de notificación del aviso, accedido mediante las funciones *SetState* y *GetState*, permitiendo conocer si el aviso a sido notificado o no.

Por otro lado la clase implementa un conjunto de funciones, *Open*, *Close* y *Play*, destinadas al control de la reproducción del mensaje del aviso a través de la llamada realizada a tal efecto.

La función *Open* es la encargada de inicializar el proceso de envío del mensaje, siendo el método llamado por el control de avisos cuando se realiza la notificación y recibiendo el objeto de control de la llamada empleada, referencia al objeto CCall empleado por CControlAverias para realizar la llamada. Dicha función realiza una llamada al método *Play* que es el encargado de realizar efectivamente la reproducción del mensaje a través de la llamada dada. Cabe destacar el hecho de que el método *OnWaveOutDone* de la clase, que como ya se ha explicado es el receptor del evento de finalización de la reproducción del fichero de voz[1], controla el número de veces que se reproduce el mensaje, dando por finalizado el proceso de notificación alcanzado un cierto número de reproducciones o volviendo a reproducir el mensaje en caso de no haberse alcanzado tal número.

[1] Véase el apartado 6.7 Clases de reproducción de voz.

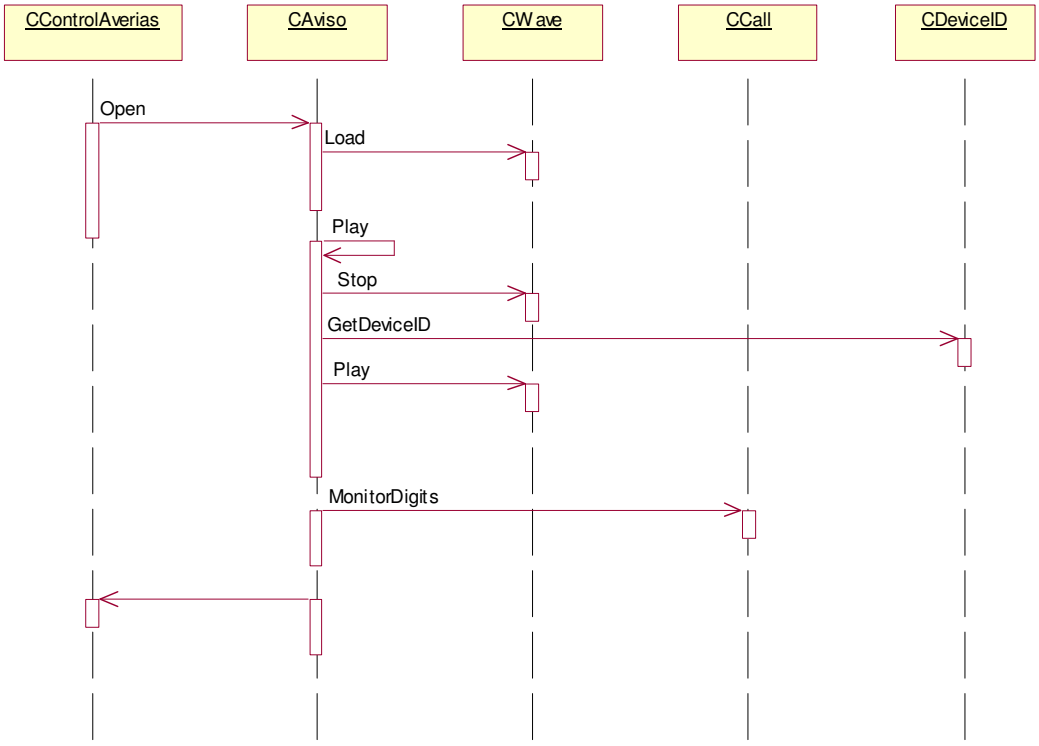

Ilustración 6-5

Por último el método *Close* da por finalizada la reproducción del aviso, liberando los recurso ocupados durante el proceso, y es llamada por la clase de control de los avisos durante la finalización del proceso de notificación.

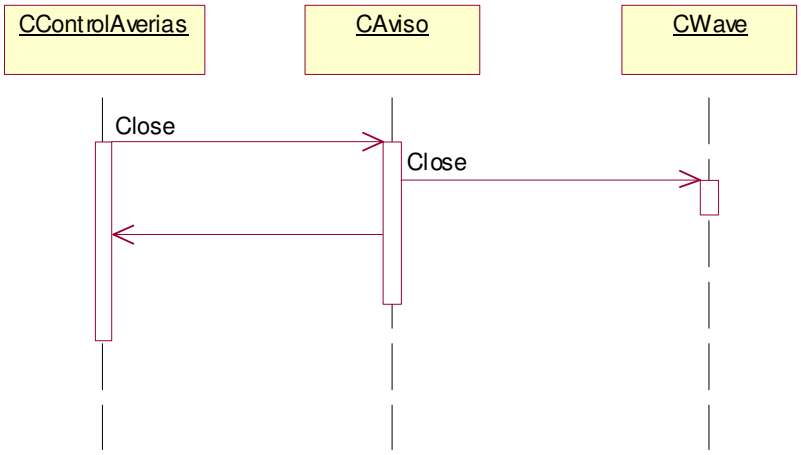

Ilustración 6-6

6.6 Clases de temporización

Bajo esta clasificación se agrupan aquellas clases que implementan el soporte necesario para el desarrollo de temporizadores y de receptores de eventos de los mismos.

Su funcionalidad se basa en el establecimiento de una cuenta, especificada en milisegundos, y que se reinicia cada vez que se alcanza el valor establecido. En este momento el sistema ejecuta una función dada como parámetro al establecerse el contador, desde la cual se puede redirigir el evento de fin de cuenta hacia el objeto de recepción de eventos del contador establecido.

De este modo se desarrollan dos clases, una para el control de los contadores propiamente dichos y otra para la implementación de los receptores de eventos. Por otro lado se establece el procedimiento que será llamado cuando se produzca el fin de cuenta, método que en nuestro caso es el llamado *TimerProc*.

6.6.1 CTimer

Es la clase encargada de la implementación de los objetos de control de los contadores. Su principal funcionalidad consiste en proporcionar métodos para la creación y gestión de los contadores. Para tal fin la clase implementa un juego de tres métodos, *Start*, *Stop* y *Running*, que permitirán a las clases que hagan uso de su funcionalidad el crear, determinado el tiempo de cuenta del contador, parar y comprobar el estado de actividad del contador, respectivamente.

El método *Start* crea y pone en marcha el contador, indicándole el procedimiento que se ejecutará cada vez que se produzca el fin de cuenta. En dicho procedimiento se recupera el objeto de la clase CTimer asociado al contador que ha producido el fin de cuenta, llamando al método *OnTimer* de dicho objeto. Desde dicho método el objeto CTimer redirige el evento de fin de cuenta al receptor de eventos de contador que se haya registrado en él, objeto de la clase CTimerSink dado en el constructor de la clase.

Siguiendo este procedimiento se podrán establecer contadores en aquellas clases que requieran de la automatización de ciertos procesos, controlando dicha automatización con estos contadores. El procedimiento se muestra con mas detalle en la siguiente ilustración.

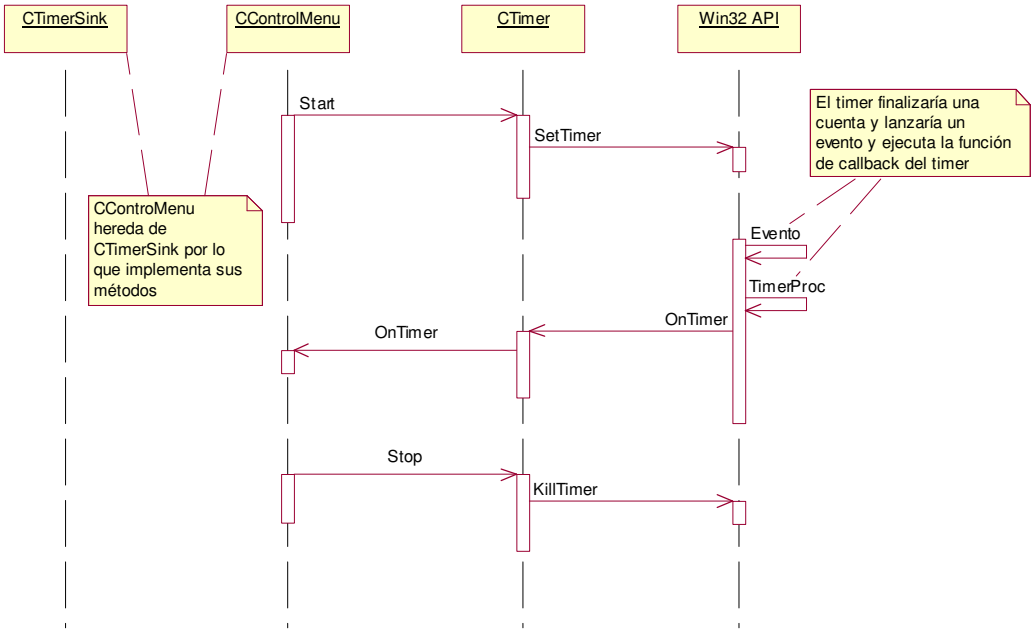

Ilustración 6-7

6.6.2 CTimerSink

Esta clase virtual, clase que solo implementa las definiciones de sus métodos pero que no desarrolla funcionalidad alguna para los mismos, implementa el mecanismo básico de comunicación de eventos de contador. Así define un método, *OnTimer*, que será llamado, tal y como se ha visto en el apartado anterior, desde el método de recepción de eventos del correspondiente controlador del contador.

De este modo derivando las clases que deban implementar la funcionalidad de los contadores de esta clase y registrándolas en sus respectivos controladores de contadores, podrán recibir los eventos de fin de cuenta que se produzcan en el objeto de la clase CTimer asociado.

6.7 Clases de reproducción de voz

En este conjunto se agrupan aquellas clases necesarias para la reproducción de los mensajes de voz a través de la línea telefónica. Implementarán los procedimientos de carga de los mensajes de voz, tanto desde el fichero de extensión como desde los propios ficheros de voz, y los mecanismos para la reproducción de los mensajes cargados.

El sistema de reproducción de mensajes de voz se basa en una librería especializada en el manejo de recursos multimedia, WINMM, que ofrece todo un conjunto de funciones y métodos para el manejo y gestión tanto de dispositivos multimedia como de ficheros de voz. Al igual que en el apartado anterior el control de eventos, tales como la finalización de la reproducción de un mensaje, se realiza a través de una función de callback indicada al comenzarse la reproducción y que se ejecuta cada vez que se produce un evento, función que en nuestro caso es la denominada *WaveOutProc*.

Así se desarrollan dos clases, una para el control de los procedimientos de carga y reproducción de los mensajes; y otra para la implementación del procedimiento de redireccionamiento de los eventos que se produzcan al reproducir los mensajes.

6.7.1 CWave

Esta clase es la encargada de realizar las operaciones requeridas para la carga y reproducción de los mensajes de voz.

Dado que mensajes de voz proceden de dos fuentes distintas, los básicos para el funcionamiento del interfaz se localizan en el fichero de extensión de sonidos, y los correspondientes a los mensajes de información y avisos, configurables por el usuario, se encuentran en ficheros de sonido, se requerían de dos procedimientos distintos para la carga de los mensajes. Así la clase tiene una función de inicialización, *Load*, que carga el mensajes de voz indicado, ya sea pasándole el nombre del fichero de sonido a cargar o pasándole el

identificador del mensajes dentro del fichero de extensión. Una vez cargado el mensaje de voz se reproduce mediante la ejecución del método *Play* al cual se le indica el identificador del dispositivo de salida de la llamada en curso, identificador obtenido mediante la clase CDeviceID[1]. Finalmente implementa dos métodos adicionales, *Stop* para detener la reproducción, y *Close* que descarga el fichero de sonido cargado y libera los recursos ocupados.

6.7.2 CWaveSink

Esta clase virtual da el soporte necesario para la comunicación de los eventos producidos durante la reproducción de los mensajes a las clases que empleen estos recursos. Así estas clases heredarán de CWaveSink y se registraran en el correspondiente objeto de la clase CWave que empleen para reproducir los mensajes. Al producirse un evento este será redirigido al CWaveSink registrado en el CWave que haya producido dicho evento, y alcanzará de este modo a la clase que estaba reproduciendo el mensaje y que debe capturar dichos eventos para actuar en consecuencia.

Los métodos que implementa esta clase son seis, tres asociados a los eventos producidos durante la reproducción, eventos de salida, y otros tres asociados a la recepción, eventos de entrada. Así tenemos:

- **OnWaveOutClose:** Asociado al evento WOM_CLOSE, producido cuando el dispositivo de salida empleado es cerrado.
- **OnWaveOutDone:** Cuyo evento asociado, WOM_DONE, se produce cuando se finaliza la reproducción.
- **OnWaveOutOpen:** Asociado al evento WOM_OPEN que se envía cuando el dispositivo de salida es abierto.
- **OnWaveInClose:** Asociado al evento WIM_CLOSE de cierre del dispositivo de entrada.

[1] Véase el apartado 6.3.6 CDeviceID

- **OnWaveInData:** Cuyo evento, WIM_DATA, es enviado cuando se reciben datos en el dispositivo de entrada.
- **OnWaveInOpen:** Llamado cuando se recibe en evento WIM_OPEN de apertura del dispositivo de entrada.

7 Módulo de sonidos: WaveLib

Como ya se ha comentado, los ficheros de sonido necesarios para el correcto funcionamiento del interfaz han sido encapsulados bajo un modulo independiente al del interfaz propiamente dicho. Este mecanismo nos permitirá modificar dichos ficheros de voz, y por lo tanto los mensajes que se reproduzcan a través del teléfono, sin que se tenga que rehacer o reconfigurar el modulo principal del interfaz.

El modulo de sonidos, llamado WaveLib, se ha generado como un fichero DLL donde se han incluido los ficheros de sonido anteriormente comentados, de tal forma que las clases que forman el interfaz puedan acceder a ellos a través de un identificador único. Dicho identificador permite distinguir entre los diversos ficheros de voz, de tal forma que cada clase accede solo aquellos que son usados por ella. La relación de identificadores únicos y el contenido del correspondiente fichero de voz se muestra en la siguiente lista:

IDR_WAVE_MENU_INICIO	Mensaje del menú de inicio.
IDR_WAVE_MENU_ACCESO	Mensaje del menú de acceso.
IDR_WAVE_MENU_ACCION	Mensaje del menú de acción.
IDR_WAVE_MENU_INFORMACION	Mensaje del menú de información.
IDR_WAVE_MENU_OPCIONES	Mensaje del menú de opciones.
IDR_WAVE_ERROR	Mensaje de error en el código del sistema introducido.
IDR_WAVE_NO_SUCCESS	Mensaje de indicación de la no finalización correcta de la operación solicitada.
IDR_WAVE_SUCCESS	Mensaje de indicación de la correcta finalización de la operación solicitada.
IDR_WAVE_START_INFORMACION	Mensaje de comienzo de la información sobre un elemento del sistema.
IDR_WAVE_ERROR_INFORMACION	Menaje de no disposición de la información solicitada.
IDR_WAVE_ACTIVATE_SINGLE	Mensaje de reproducción del estado de activado.
IDR_WAVE_ACTIVATE_NO_SINGLE	Mensaje de reproducción del estado de activados.

IDR_WAVE_NO_ACTIVATE_SINGLE	Mensaje de reproducción del estado de no activado.
IDR_WAVE_NO_ACTIVATE_NO_SINGLE	Mensaje de reproducción del estado de no activados.
IDR_WAVE_NON_ACTIVATE_SINGLE	Mensaje de reproducción del estado de encendido.
IDR_WAVE_NON_NO_ACTIVATE_SINGLE	Mensaje de reproducción del estado de apagado.
IDR_WAVE_NON_ACTIVATE_NO_SINGLE	Mensaje de reproducción del estado de encendidos.
IDR_WAVE_NON_NO_ACTIVATE_NO_SINGLE	Mensaje de reproducción del estado de apagados.
IDR_WAVE_UNO	Mensaje de reproducción del valor uno.
IDR_WAVE_DOS	Mensaje de reproducción del valor dos.
IDR_WAVE_TRES	Mensaje de reproducción del valor tres.
IDR_WAVE_CUATRO	Mensaje de reproducción del valor cuatro.
IDR_WAVE_CINCO	Mensaje de reproducción del valor cinco.
IDR_WAVE_SEIS	Mensaje de reproducción del valor seis.
IDR_WAVE_SIETE	Mensaje de reproducción del valor siete.
IDR_WAVE_OCHO	Mensaje de reproducción del valor ocho.
IDR_WAVE_NUEVE	Mensaje de reproducción del valor nueve.
IDR_WAVE_DIEZ	Mensaje de reproducción del valor diez.
IDR_WAVE_ONCE	Mensaje de reproducción del valor once.
IDR_WAVE_DOCE	Mensaje de reproducción del valor doce.
IDR_WAVE_TRECE	Mensaje de reproducción del valor trece.
IDR_WAVE_CATORCE	Mensaje de reproducción del valor catorce.
IDR_WAVE_QUINCE	Mensaje de reproducción del valor quince.
IDR_WAVE_DIECISEIS	Mensaje de reproducción del valor dieciséis.
IDR_WAVE_DIECISIETE	Mensaje de reproducción del valor diecisiete.
IDR_WAVE_DIECIOCHO	Mensaje de reproducción del valor dieciocho.
IDR_WAVE_DIECINUEVE	Mensaje de reproducción del valor diecinueve.
IDR_WAVE_VEINTE	Mensaje de reproducción del valor veinte.
IDR_WAVE_TREINTA	Mensaje de reproducción del valor treinta.
IDR_WAVE_CUARENTA	Mensaje de reproducción del valor cuarenta.
IDR_WAVE_CINCUENTA	Mensaje de reproducción del valor cincuenta.
IDR_WAVE_SESENTA	Mensaje de reproducción del valor sesenta.
IDR_WAVE_SETENTA	Mensaje de reproducción del valor setenta.
IDR_WAVE_OCHENTA	Mensaje de reproducción del valor ochenta.
IDR_WAVE_NOVENTA	Mensaje de reproducción del valor noventa.
IDR_WAVE_CIEN	Mensaje de reproducción del valor cien.
IDR_WAVE_CIENTOS	Mensaje de reproducción de cientos.
IDR_WAVE_COMA	Mensaje de reproducción de la coma decimal.

El proceso de carga de los ficheros de sonido se lleva a cabo en un proceso que sigue tres pasos:

1. Se carga el módulo WaveLib.
2. Se carga el fichero de sonido deseado a partir del identificador que lo identifica.

3. Se cierra el módulo de sonidos.

El proceso se muestra con más detalle en la siguiente ilustración:

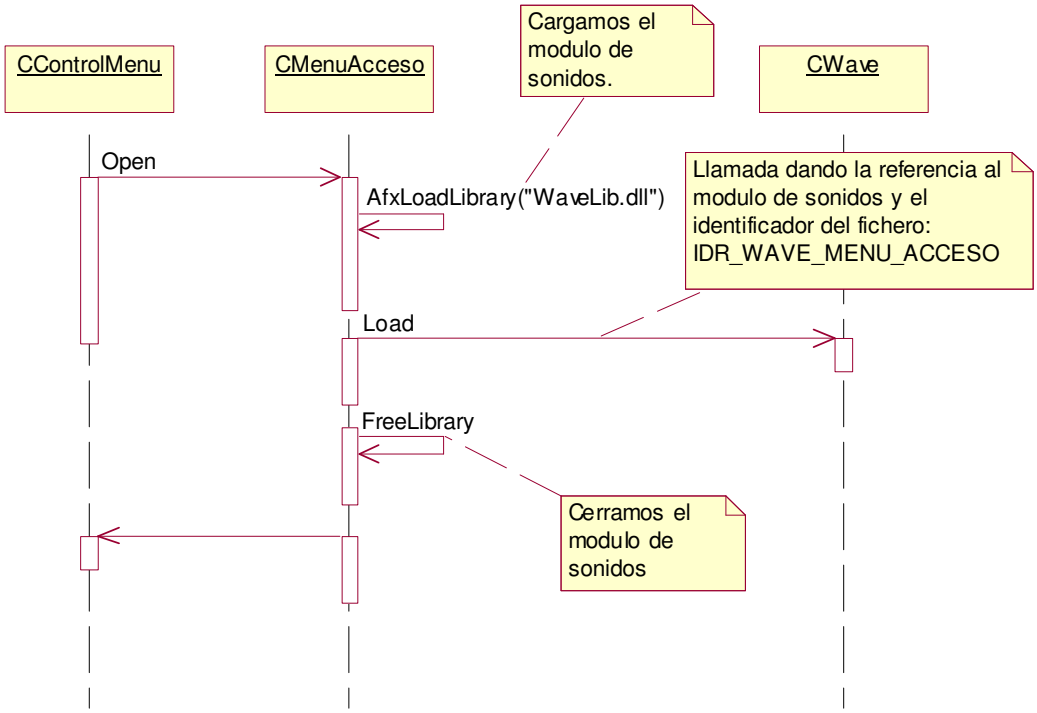

Ilustración 7-1

El proceso de carga del módulo nos proporciona una referencia al mismo que es dada a la clase CWave a través de su método Load. Dicho método carga el recurso de sonido identificado en el segundo de sus parámetros, en el caso de la ilustración el fichero del menú de acceso, desde el módulo especificado. Dicha referencia al módulo de sonidos nos permite cerrarlo en el último paso del proceso.

8 Anexo 1: TAPI

Como ya se ha comentado bajo el nombre de TAPI se encuentra una extensísima API, Application Programming Interface, que ofrece un método estandarizado de acceso a los recursos telefónicos, permitiendo de este modo el acceso a duchos recursos desde las aplicaciones informáticas que los requieran y posibilitando la conexión de los sistemas informáticos con las líneas telefónicas.

TAPI apareció por primera vez en Mayo de 1993 en su versión 1.0 y desde entonces ha ido integrada con las distintas versiones de Windows que han sido desarrolladas. Así la versión 1.3 de TAPI es soportada ya en el Windows 3.11, del mismo modo que la versión 1.4 lo esta en el Windows 95 y la 2.0 en el Windows NT4. Los más modernos sistemas operativos Windows implementan la última versión de esta potente herramienta, la 3.0, que incluye algunos de las más modernas tecnologías informáticas, tales como COM y la programación orientada a objetos. Incluso aparece implementado en el Windows CE desarrollado para los ordenadores de mano. Esta prolongación en el tiempo de TAPI da una idea de su amplitud y versatilidad, así de como su potencia. TAPI ofrece funciones y mecanismo para dar soporte no solo a aplicaciones telefónicas más sencillas y comunes, tales como contestadores automáticos o faxes, sino que además permite un amplísimo abanico de distintos tipos de conexiones, como ya veremos más adelante, y funcionalidades mucho mas complejas, pudiéndose llegar a implementar auténticas centralitas telefónicas o CallCenter.

La arquitectura básica de TAPI es la que se muestra en la siguiente figura:

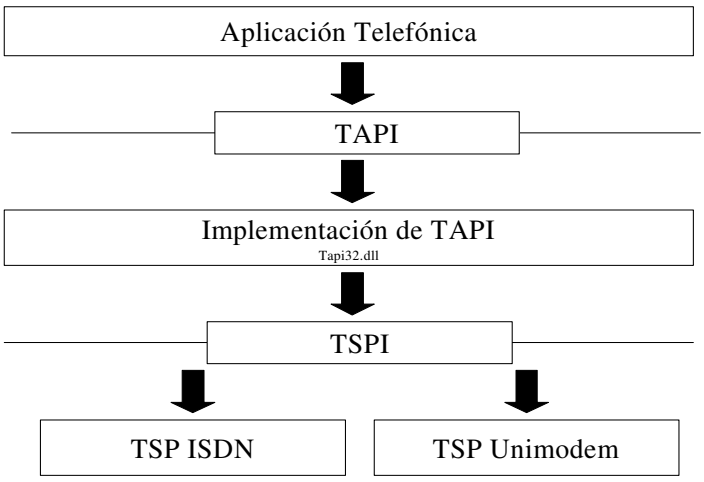

Ilustración 8-1

Como puede apreciarse TAPI actúa como interfaz entre las aplicaciones informáticas y los distintos recursos telefónicos haciendo uso a su vez de un segundo interfaz de estandarización, el TSPI, Telephony Service Provider Interface. Este segundo interfaz proporciona un conjunto de métodos estandarizados de acceso a los distintos servicios que ofrecen los proveedores de servicios telefónicos o TSPs[1]. De este modo los distintos TSPs implementarían las funciones del TSPI haciendo uso de sus propios recursos y ajustados a su hardware, obteniéndose así una total independencia entre el software de la aplicación informática y el hardware empleado para acceder a las funcionalidades telefónicas: El desarrollador de la aplicación no debe preocuparse por el hardware de cada máquina donde se instale su aplicación, y los TSP desarrollar su hardware independientemente de las aplicaciones telefónicas existentes.

Esta arquitectura basada en capas, o interfaces, proporciona a TAPI una gran versatilidad e independencia. No depende del TSP, tal y como hemos visto, lo que le permite poder ser usada para acceder a cualquier tipo de red de telefonía, analógica, ISDN, móvil. Los distintos protocolos de comunicaciones que usan estas redes pueden ser accedidos por TAPI, siempre que el TSP desarrolle su parte del TSPI. Esto permite el acceso a las distintas funcionalidades

[1] No se trata de los distintos operadores de telefonía, sino de los fabricantes de hardware de acceso a los recursos de dichos operadores, es decir, los fabricantes de modems analógicos, digitales o GSM.

desarrolladas por cada una de ellas, aunque hay que indicar que las funcionalidades disponibles en cada caso dependen del TSP al que se acceda. Es responsabilidad de los desarrolladores el emplear las funciones adecuadas al tipo de TSP que en cada momento se esté empleando, aunque cabe destacar que TAPI permite conocer dichas funciones y ajustar la funcionalidad de la aplicación informática a las mismas.

En el presente proyecto fin de carrera hacemos uso de la versión 2.0 de TAPI por ser la más avanzada sin necesidad de emplear los más complejos sistemas de la versión 3.0. Además de esta forma no aseguramos cubrir un mayor número de posibles usuarios, ya que la versión 3.0 no es compatible con la 2.0 y solo se encuentra en los más modernos sistemas Windows, siendo la 2.0 totalmente compatible con la 3.0.

Esta versión de TAPI, la versión 2.0, se basa en un conjunto de métodos que ofrecen las distintas funcionalidades que TAPI es capaz de proveer, de forma que se empleara uno determinado cada vez que se quiera acceder a una de estas funcionalidades, por ejemplo para realizar una llamada se emplea el método lineMakeCall. Con el fin de aumentar la modularidad e independencia de las distintas funcionalidades que nuestra aplicación requiere de TAPI, se ha optado por encapsular los distintos métodos o conjuntos de métodos, en aquellos casos en los que su funcionalidad este relacionada, bajo clases. De este modo conseguimos establecer un conjunto de elementos básicos que, haciendo uso de los métodos de TAPI, ofrecen las funciones necesarias para el desarrollo de la aplicación, consiguiendo al mismo tiempo una elevada independencia de los mismos lo que aumenta sus posibilidades de reutilización y mantenimiento.

Por otro lado la gestión que hace TAPI de los recursos telefónicos a los que accede está basada en eventos. Es decir, TAPI genera distintos eventos o sucesos como respuesta a los cambios y acciones que se produzcan en la línea, tanto por su propia actividad como por las acciones que a través de TAPI se lleven a cabo sobre ella. Dichos eventos permiten determinar que suceso ha tenido lugar en la línea en el momento en el que se recibe de forma que se puede actuar conforme a dicho suceso, por lo que la aplicación que haga uso de TAPI debe de

capturar dichos eventos y procesarlos según sus propias necesidades. TAPI ofrece un mecanismo para la comunicación de estos eventos a la aplicación que lo requiera, basado en lo que se denomina función de callback. Dicha función debe ser implementa por la aplicación base siguiendo las directivas dadas por TAPI[1] y comunicada a TAPI en el momento de su inicialización. TAPI ejecutará dicha función cada vez que lance un evento, de forma que la aplicación base podrá capturar dichos eventos y actuar conforme a los mismos. El formato de dicha función de callback se muestra a continuación

```
void CALLBACK LineCallbackFunc (DWORD hDevice,
                                DWORD dwMsg,
                                DWORD wCallbackInstance,
                                DWORD dwParam1,
                                DWORD dwParam2,
                                DWORD dwParam3)
```

Manteniendo la definición básica en cuanto al número y tipo de los parámetros de la función y los modificadores del tipo de función, la función de callback implementada por la aplicación puede denominarse como el desarrollador determine. La implementación que se haga de esta función tratará los eventos generados por TAPI y la información que los acompaña según las necesidades de la aplicación. De este modo cabe destacar el significado de cada uno de los parámetros de la función de callback,:

- DWORD hDevice: Manejador del dispositivo que ha generado el suceso que ha disparado el evento. Este valor permitirá discriminar la procedencia del evento pudiéndose actuar de un modo u otro según este valor.
- DWORD dwMsg: Identificador del evento TAPI que se ha recibido[2] lo que permitirá filtrar y redirigir los eventos según su tipo.
- DWORD wCallbackInstance: Información establecida por el usuario al abrir el dispositivo que ha generado el evento. El valor de este parámetro viene

[1] Consúltese la siguiente dirección web para más detalles: http://msdn.microsoft.com/library/en-us/tapi/tapi2/linecallbackfunc.asp
[2] Para más detalles sobre loe eventos TAPI: http://msdn.microsoft.com/library/en-us/tapi/tapi2/line_device_messages.asp

determinado por el que se estableció en la correspondiente función de apertura del dispositivo[1] y es el mismo en todos los eventos que este dispositivo genere. Permite que el desarrollador intervenga en los datos que el evento pase a la aplicación.

- DWORD dwParam1, DWORD dwParam2, DWORD dwParam3: El valor y significado de estos tres parámetros depende del tipo de evento que se haya generado, completan la información que se recibe sobre el sistema y permiten un mayor grado de tratamiento de los eventos.

Existen otros dos métodos para la comunicación de eventos desde TAPI a la aplicación que hace uso de sus recursos, uno basado en eventos y otro basado en puertos de finalización, pero resultan más complejos y difíciles de emplear. En ambos casos es la aplicación base debe proporcionar a TAPI los manejadores de los eventos o puertos que TAPI empleará para comunicar los eventos, siendo su responsabilidad el crear, mantener y cerrar dichos procedimientos.

Como ya se ha comentado esta función de callback debe ser establecida en el momento en el que se inicialice TAPI. Este es otro de los pasos necesarios para hacer uso de los recursos proporcionados por TAPI, ya que de esta inicialización permite establecer la función de callback y establece la asociación necesaria entre la aplicación y TAPI. Para realizar esta inicialización se hace uso de la siguiente función[2]:

```
LONG lineInitializeEx(LPHLINEAPP lphLineApp,
                      HINSTANCE hInstance,
                      LINECALLBACK lpfnCallback,
                      LPCSTR lpszFriendlyAppName,
                      LPDWORD lpdwNumDevs,
                      LPDWORD lpdwAPIVersion,
                      LPLINEINITIALIZEEXPARAMS lpLineInitializeExParams)
```

1 Véase el apartado sobre la función lineOpen para más detalles.
2 Consúltese la página http://msdn.microsoft.com/library/en-us/tapi/tapi2/lineinitializeex.asp para más detalles.

Entre los parámetro más destacados cabe mencionar los siguientes, ya que son los que nos permitirán relacionar nuestra aplicación con los recursos manejados por TAPI:

- LPHLINEAPP lphLineApp: Se trata de un valor que nos devuelve TAPI e identifica la conexión entre nuestra aplicación y TAPI, haciendo las veces de manejador de TAPI.
- HINSTANCE hInstance: Manejador de nuestra aplicación, permite a TAPI identificar la aplicación con la que debe establecer la asociación. Se trata de un parámetro de entrada para TAPI y es básico para la correcta inicialización del mismo.
- LINECALLBACK lpfnCallback: Referencia a nuestra función de callback que implementada por nuestra aplicación y que TAPI ejecutará cuando tenga que comunicar algún evento.
- LPCSTR lpszFriendlyAppName: Cadena de caracteres con el nombre de nuestra aplicación.
- LPLINEINITIALIZEEXPARAMS lpLineInitializeExParams: Referencia a una estructura del tipo LINEINITIALIZEEXPARAMS[1] donde se establecen los parámetros que determinan el tipo de relación entre la aplicación y TAPI. Básicamente determinan el tipo de mecanismo que empleará TAPI para comunicar los eventos a la aplicación. En nuestro caso el campo dwOptions de dicha estructura deberá establecerse al valor LINEINITIALIZEEXOPTION_USEHIDDENWINDOW para hacer uso del mecanismo de callback.
- LPDWORD lpdwNumDevs: Número de dispositivos que soportan TAPI detectados en el sistema. TAPI evalúa los dispositivos conectados en el sistema y determina automáticamente cuales de ellos pueden usarse empleando los recursos de TAPI. Este chequeo se basa en la implementación

[1] Véase la página http://msdn.microsoft.com/library/en-us/tapi/tapi2/lineinitializeexparams_str.asp para más detalles.

del correspondiente TSPI para dichos dispositivos, ya que es este el que le proporciona a TAPI los mecanismos para acceder al dispositivo.

Para poder hacer uso de los recursos de TAPI, y tras llamar a la función de inicialización, habría que realizar lo que se denomina negociación de la versión de TAPI. Este último paso le permite a la aplicación establecer la versión de TAPI de la que hará uso. Dicho proceso de negociación se basa en otra función proporcionada por TAPI, lineNegotiateAPIVersion[1]. Dicha función debe ser ejecutada sobre cada uno de los dispositivos anteriormente detectados por TAPI y que se deseen emplear. Una descripción más detallada de esta función se muestra a continuación:

```
LONG lineNegotiateAPIVersion(HLINEAPP hLineApp,
                             DWORD dwDeviceID,
                             DWORD dwAPILowVersion,
                             DWORD dwAPIHighVersion,
                             LPDWORD lpdwAPIVersion,
                             LPLINEEXTENSIONID lpExtensionID);
```

Como se ha comentado esta función debe ser ejecutada sobre cada uno de los dispositivos detectados por TAPI y que se desee emplear. Entre sus parámetros determinan el rango de versiones de TAPI que se desea emplear, dándonos aquel que mejor se ajuste al dispositivo. Así tenemos los siguientes parámetros:

- HLINEAPP hLineApp: Manejador asociado a TAPI y obtenido en la inicialización de TAPI, permite determinar sobre que asociación de TAPI se está trabajando.
- DWORD dwDeviceID: Identificador del dispositivo sobre el que se desea actuar, normalmente un valor entre el 0 y el número de dispositivos detectados por TAPI en la llamada a la función de inicialización de TAPI.
- DWORD dwAPILowVersion: Versión más baja de TAPI que se desea emplear.

[1] Consúltese la página http://msdn.microsoft.com/library/en-us/tapi/tapi2/linenegotiateapiversion.asp para más detalles.

- DWORD dwAPIHighVersion: Versión más alta de TAPI que se desea emplar.
- LPDWORD lpdwAPIVersion: Referencia a la variable en la que TAPI almacenará la versión de TAPI negociada con el dispositivo, en caso de que dicha negociación se realice correctamente.

Tras realizar correctamente la negociación de la versión de TAPI y para poder identificar los dispositivos sobre los que podemos establecer llamadas de voz, con el fin de evitar emplear dispositivos que no soporten este tipo de conexiones y el sistema falle, se deben obtener las capacidades de los dispositivos que se deseen emplear mediante la llamada a la función[1] de TAPI lineGetDevCaps[2] con el fin de filtrar solo aquellos que cumplan ciertas condiciones. Dicha función está sujeta a la siguiente definición:

```
LONG lineGetDevCaps(HLINEAPP hLineApp,
                    DWORD dwDeviceID,
                    DWORD dwAPIVersion,
                    DWORD dwExtVersion,
                    LPLINEDEVCAPS lpLineDevCaps);
```

El parámetro más significativo es sin duda alguna el último de los parámetros de la definición de la función. Se trata de una referencia a una estructura del tipo LINEDEVCAPS[3]. Esta estructura, al igual que prácticamente la totalidad de las estructuras de datos empleadas por TAPI, no tiene una longitud fija, sino que su longitud, la cantidad de memoria que ocupa, depende del momento en el que se ejecute la función y el dispositivo sobre el que se ejecute. Esto obliga a realizar un proceso de continuas llamadas a la función hasta que esta informe que las dimensiones de la estructura han sido lo suficientemente grades como para almacenar todos los datos disponibles, tal y como puede apreciarse en la implementación del método *GetDevCaps* de la clase *CLineDevCaps*, consúltese el código fuente para más detalles.

Así tenemos los siguientes parámetros en la llamada a la función:

[1] En nuestro caso dicha llamada se hace a través de la clase CLineDevCaps, véase el apartado 6.3.7 CLineDevCaps.
[2] http://msdn.microsoft.com/library/en-us/tapi/tapi2/linegetdevcaps.asp para más información sobre esta función.
[3] Consúltese la siguiente página web para más detalles: http://msdn.microsoft.com/library/en-us/tapi/tapi2/linedevcaps_str.asp.

- HLINEAPP hLineApp: Manejador de la conexión a TAPI sobre la que se actua.
- DWORD dwDeviceID: Identificador del dispositivo sobre el que se desea obtener sus capacidades.
- DWORD dwAPIVersion: Versión de TAPI empleada para actuar sobre el dispositivo.
- DWORD dwExtVersion: Puesto a cero pues es un dato que para las pretensiones del actual proyecto fin de carrera no es necesario.
- LPLINEDEVCAPS lpLineDevCaps: Referencia a la estructura LINEDEVCAPS que TAPI rellenará con los datos del dispositivo.

Para establecer que el dispositivo soporta el tipo de llamadas necesario para la funcionalidad requerida, deben darse las siguientes tres condiciones:

- El parámetro dwBearerModes de la estructura LINEDEVCAPS debe incluir el identificador LINEBEARERMODE_VOICE, lo que indica que soporta llamadas entrantes de voz.
- El parámetro dwMediaModes debe incluir el identificador LINEMEDIAMODE_AUTOMATEDVOICE, lo que indica que el dispositivo es capaz de soportar llamadas de voz automáticas, es decir, la propia aplicación es capaz de responder a la llamada por si sola, sin intervención exterior.
- Por último el parámetro dwLineFeatures debe incluir el valor LINEFEATURE_MAKECALL, lo que indica que le dispositivo puede realizar llamadas.

Solo se emplearan aquellos dispositivos detectados que hayan negociado correctamente la versión de TAPI y que soporten los tres modos anteriores.

Llegados a este punto abríamos inicializado TAPI para su posterior uso e identificado los dispositivos que pueden ser configurados y empleados por nuestra aplicación. A partir de este

punto la aplicación podrá abrir líneas para quedar a la escucha de las llamadas entrantes y realizar llamadas sobre dicha línea. Para ello se hace uso de otra función de TAPI, enmascarada en nuestro caso por la clase *CLine*[1], llamada lineOpen[2] encargada de abrir una línea sobre la conexión a TAPI. Dicha función está definida tal y como se muestra a continuación:

```
LONG lineOpen(HLINEAPP hLineApp,
              DWORD dwDeviceID,
              LPHLINE lphLine,
              DWORD dwAPIVersion,
              DWORD dwExtVersion,
              DWORD dwCallbackInstance,
              DWORD dwPrivileges,
              DWORD dwMediaModes,
              LPLINECALLPARAMS const lpCallParams);
```

Así los parámetros más destacados por tener una mayor influencia en el comportamiento de TAPI a la hora de gestionar la línea abierta y los eventos que en ellas se producen son:

- HLINEAPP hLineApp: Manejador de la conexión a TAPI sobre la que se abrirá la línea, identificando las características básicas de la gestión de la línea según las propiedades establecidas al abrir la conexión.
- DWORD dwDeviceID: Identificador del dispositivo sobre el cual se establecerá la línea abierta, condicionando según sus capacidades las de la propia línea.
- LPHLINE lphLine: Referencia a una variable del tipo HLINE donde TAPI almacenará el manejador único que identifica a la línea abierta, en caso de que dicha apertura se realice correctamente.
- DWORD dwCallbackInstance: Los datos dados en este parámetro serán devueltos por TAPI en el parámetro wCallbackInstance de la función de

[1] Consúltese el apartado 6.2.4 CLine para más detalles de este clase.
[2] Véase la pagina http://msdn.microsoft.com/library/library/en-us/tapi/tapi2/lineopen.asp para más detalles.

callback establecida al inicializar la conexión con TAPI, cada vez que comunique un evento asociado a la línea abierta. Se trata de una variable de 32 bits, por lo que puede ser dado un dato determinado o una referencia a algún objeto o estructura de datos.

- DWORD dwPrivileges: Según su valor se determinan los tipos de llamadas que la aplicación que abre la línea puede gestionar sobre la misma.
- DWORD dwMediaModes: Determina el tipo de llamadas que la aplicación podrá gestionar sobre la línea abierta.

Abierta la línea la aplicación podrá pasar a recibir llamadas o realizarlas. Para ello empleará un par de métodos proporcionados por TAPI, que en nuestro caso quedan enmascaradas por la clase CCall[1]. Dichos métodos forman la base de la gestión de las llamadas que hace TAPI, aunque son una pequeña parte de todas las que TAPI ofrece en este campo. Así la realización de las llamadas se basa en la función lineMakeCall[2] que se define de la siguiente forma:

```
LONG lineMakeCall(HLINE hLine,
                  LPHCALL lphCall,
                  LPCSTR lpszDestAddress,
                  DWORD dwCountryCode,
                  LPLINECALLPARAMS const lpCallParams);
```

Como se ha comentado en nuestro caso dicha función queda enmascarada por el método *MakeCall* de la clase CCall. Para la realización de la llamada se requiere identificar la línea que dará soporte a la llamada así como los parámetros necesarios para realizar físicamente la llamada, así los parámetros de esta función son los siguientes:

- HLINE hLine: Manejador de la línea, previamente abierta, sobre la que se realizará la llamada.

[1] Véase el apartado 6.2.5 CCall para más detalles sobre esta clase.
[2] Consúltese la siguiente dirección web para más detalles http://msdn.microsoft.com/library/en-us/tapi/tapi2/linemakecall.asp.

- LPHCALL lphCall: Referencia a una variable del tipo HCALL, donde TAPI almacenará el manejador único que nos servirá para identificar la llamada.
- LPCSTR lpszDestAddress: Cadena de caracteres con el número de teléfono, en formato marcable, al que se desea llamar.
- DWORD dwCountryCode: Código del país donde se localiza el número anterior. Si se establece a 0 TAPI toma el que este definido por defecto en las propiedades de marcado, accesibles como ya se comento a través de la función lineTranslateDialog enmascarada por el método ConfigDialing de la clase CControlMenu.
- LPLINECALLPARAMS const lpCallParams: Referencia a una estructura del tipo LINECALLPARAMS. La información contenida en esta estructura permite definir más finamente las características de la llamada, aunque en nuestro caso no se empleará tomando TAPI la configuración por defecto.

Por otro lado la recepción de las llamadas entrantes se basa en la función lineAnswer[1], que se ajusta a la siguiente definición:

```
LONG lineAnswer(HCALL hCall,
                LPCSTR lpsUserUserInfo,
                DWORD dwSize);
```

En este caso el manejador de la llamada no nos lo proporciona la ejecución de esta función sino que debe ser conocido de antemano. Esto se debe a que la función actúa sobre una llamada ya identificada y localizada por TAPI. Este proceso de identificación y localización se realiza automáticamente cada vez que TAPI detecta una nueva llamada entrante y lanza el correspondiente mensaje, LINE_NEWCALL. En este momento TAPI localiza la llamada y le asigna un manejador que es dado a la función de callback a través del parámetro dwParam1. Así recogiendo como manejador de la llamada entrante este valor al recibir el correspondiente evento podremos llamar a la función de recepción de llamadas. Sus parámetros son:

[1] Véase la web http://msdn.microsoft.com/library/library/en-us/tapi/tapi2/lineanswer.asp para más detalles.

- HCALL hCall: Manejador de la llamada que se desea recepcionar.
- LPCSTR lpsUserUserInfo: Cadena de caracteres con la información que será enviada al llamante en el momento en el que la llamadas sea físicamente respondida. Si no se especifica TAPI interpreta que no se desea enviar información alguna.
- DWORD dwSize: Tamaño en bytes de la información anterior.

Empleando el conjunto de funciones comentadas podría accederse a la funcionalidad básica que TAPI ofrece para la recepción y realización de llamadas telefónicas. Otras funcionalidades más complejas, como la recepción de las pulsaciones de las teclas o el proceso de finalización de las llamadas, siguen las mismas directivas que los casos comentados: Se basan en la recepción del correspondiente evento TAPI y en la ejecución de la función de TAPI desarrollada para dar soporte a esa funcionalidad. Por ejemplo para la habilitación de la recepción de las pulsaciones de las teclas se emplearía la siguiente función[1]:

```
LONG lineMonitorDigits(HCALL hCall,
                       DWORD dwDigitModes);
```

Dicha función habilita la recepción de las pulsaciones de teclas en la llamada identificada por el manejador de llamada, primer parámetro, y que se ajusten al modo establecido en el segundo parámetro:

- HCALL hCall: Manejador de la llamada en la que se habilitará la recepción de las pulsaciones de las teclas.
- DWORD dwDigitModes: Modo o modos que establecen las teclas detectadas. Puede ser uno o varios de los siguientes valores:

[1] Consúltese la página web http://msdn.microsoft.com/library/library/en-us/tapi/tapi2/linemonitordigits.asp para más detalles.

- LINEDIGITMODE_PULSE: Solo se detectan pulsaciones de las teclas que van del '1' al '9'. Usado para los teléfonos de marcación por pulsos.
- LINEDIGITMODE_DTMF: Se detectan las teclas del '1' al '9' y las teclas '#' y '*'. Empleado para los teléfonos de marcación por tonos.
- LINEDIGITMODE_DTMFEND: Igual que el anterior solo que además detecta las teclas 'A', 'B', 'C' y 'D'.
- O establece a nulo, 0, en cuyo caso indica que se deshabilita la detección de las pulsaciones de teclas.

Establecida la habilitación de la pulsación de las teclas cada vez que se pulse una de ellas en el teléfono llamante TAPI genera el evento LINE_MONITORDIGITS. Dicho evento sería recibido en la función de callback indicándose la tecla pulsada en el parámetro dwParam1 y el modo de la misma en el dwParam2. De esta forma ejecutando la función comentada y tratando adecuadamente el correspondiente evento TAPI, nuestra aplicación puede detectar y gestionar las pulsaciones de teclas que se produzcan en el teléfono remoto.

9 Anexo 2: Voz

El sistema planteado se basa en las pulsaciones de las teclas del teléfono desde el que se realiza el acceso para la actuación sobre el sistema y en la reproducción de ficheros de voz pregrabados para la comunicación de mensajes desde el sistema al usuario. Una de las opciones o posibles ampliaciones del presente proyecto fin de carrera radicaría en el empleo de motores de reconocimiento de voz y de conversión de texto a voz para la implementación del interfaz.

Los motores de reconocimiento de voz, denominados SR[1], permiten la transformación de la voz a texto escrito mediante el análisis de la señal de voz. Básicamente se basan en el reconocimiento de los fonemas pronunciados en la señal de voz y su conversión a sus correspondientes sílabas. De este modo forman el texto que representa las palabras y frases pronunciadas en la señal de voz. Su introducción en el presente sistema permitiría prescindir de las pulsaciones de las teclas como mecanismo actuación del usuario sobre el sistema de control domótico. Dicho mecanismo sería sustituido por uno mucho más cómodo para el usuario basado en la pronunciación de la opción deseada, en el caso de los menús basados en opciones, o en la dicción del nombre del sistema sobre el que se desea actuar o pedir información. La introducción del motor SR permitiría al convertir la voz del usuario, recibida a través del teléfono, a texto que luego sería procesado para realizar la correspondiente actuación.

En el caso de los motores de conversión de texto a voz, motores TTS[2], su empleo no iría destinado tanto a la mejora del sistema de comunicación entre el usuario y el sistema, sino a la flexibilización del mecanismo de comunicación de información del interfaz al usuario. En el sistema actual este mecanismo se basa en la reproducción de mensajes de voz pregrabados que son reproducidos según el estado las necesidades del sistema. Aunque los mensajes básicos para el funcionamiento del interfaz han sido encapsulados de forma que resultan del todo transparentes para el usuario y el desarrollador de la aplicación base, algunos otros, no

[1] Del ingles Speech Recognition.
[2] Del ingles Text To Speech.

necesarios para el correcto funcionamiento del sistema, deben ser mantenidos por el usuario final en caso de querer disponer de los mismos. El empleo de los motores TTS permitiría suprimir estos mensajes y sustituirlos por textos adecuados a cada una de las situaciones del sistema. De este modo la necesidad de grabar y mantener los mensajes de voz sería sustituida por una mucho más fácil y cómoda introducción de textos, pero sobre todo al usuario final y al desarrollador de la aplicación base se les proporcionaría un sistema mucho más flexible a la hora de personalizar los mensajes de información que el sistema emplearía a la hora de comunicar información.

Una de las herramientas disponibles en la actualidad para el desarrollo tanto de motores SR como TTS es la API(Application Programming Interface) desarrollada por Microsoft a tal efecto, SAPI. Dicha API, al igual que la ya comentada TAPI para el manejo de los recursos telefónicos, es de libre distribución pudiéndose obtener desde las pagina web de Microsoft para el desarrollo tanto de los motores propiamente dichos como para el desarrollo de aplicaciones que hagan uso de los mismos, y resulta totalmente compatible con los sistemas operativos Windows. Estas razones la hacen muy aconsejable para su empleo en el presente proyecto fin de carrera ya que, al igual que TAPI, no suponen una carga económica significativa y evitan posibles incompatibilidades con el resto del sistema.

SAPI ofrece, como ya se ha comentado, funciones tanto para el desarrollo de motores SR y TTS como para el desarrollo de aplicaciones que hagan uso de los mismos. SAPI está estructurado como un conjunto de objetos COM, organizados por áreas de funcionalidad. Su arquitectura se basa en el siguiente esquema.

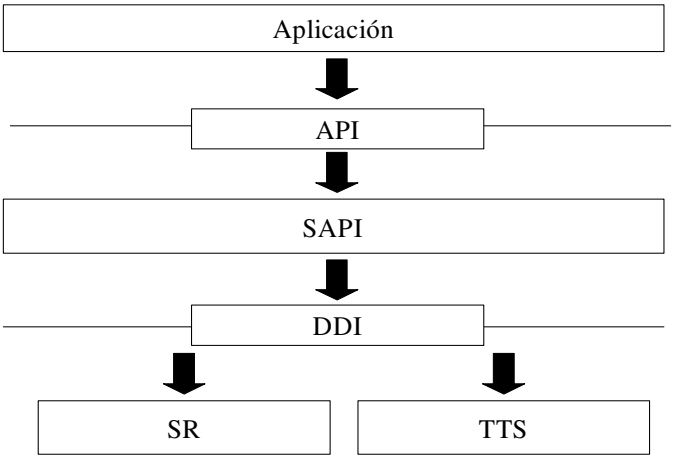

Ilustración 9-1

De este modo los objetos COM de SAPI se organizan en dos grandes grupos:

- Los destinados a implementar el interfaz con las aplicaciones que requieran hacer uso de las funcionalidades dadas por SAPI, los denominados objetos o interfaces de aplicación, Application Level Interfaces. Implementan lo que en la ilustración 9-1 se denomina API.
- Los destinados a la implementación de los motores en si mismos y a servir de pasarela entre los motores y las funcionalidades dadas por SAPI, denominados objetos o interfaces de motor, Engine Level Interfaces. Implementan la capa denominada DDI en la ilustración 9-1.

Dentro de cada uno de estos dos grandes grupos, los objetos SAPI se subdividen en otros más especializados según cual sea su funcionalidad. Así por ejemplo dentro de los objetos de aplicación se distinguen entre objetos de audio, de TTS o de SR, entre otros.

El estudio de los motores de reconocimiento y generación de voz y su desarrollo ocuparía más de lo que se pretende en el presente proyecto fin de carrera, constituyéndose por si solos en un tema propio de todo un nuevo proyecto, pero intentaremos dar una idea básica de cómo se podría hacer uso de la herramienta comentada para realizar dicho trabajo.

SAPI ofrece cuatro objetos básicos para el empleo y desarrollo de los motores TTS y SR, uno para cada motor y cada grupo de objetos, de forma que el trabajo en cada caso giraría en torno a uno de estos objetos según sea la finalidad que buscamos:

- Así si lo que se pretende en dar soporte de reconocimiento de voz a una aplicación a partir de un motor SR ya desarrollado, la aplicación contará con el soporte dado por SAPI a partir del objeto ISpRecognizer[1]. Este objeto de SAPI permitirá a la aplicación controlar el motor de reconocimiento de voz, creando uno por cada motor que desee emplear. A partir de este objeto, SAPI implementa otros destinados a obtener y manejar la información que del anterior se obtenga, como por ejemplo el objeto ISpRecoResult[2] para obtener las hipótesis.
- Si por el contrario se desea emplear un motor TTS para dar soporte de conversión de texto voz a una aplicación, se hará uso del objeto ISpVoice[3]. Dicho objeto permitirá a la aplicación hacer uso de un motor TTS previamente desarrollado y poder hacer uso de sus funciones de conversión de texto o voz.
- Si se desea implementar un motor SR compatible con SAPI, dicho motor deberá realizarse en torno a un objeto del tipo ISpSREngine[4]. Dicho objeto es empleado por SAPI a través del DDI para hacer uso de las funcionalidades del motor, por lo que debe ser el elemento central del motor que deseamos desarrollar.
- Del mismo modo si se desea desarrollar un motor TTS, este deberá implementar un objeto del tipo ISpTTSEngine[5], lo que le permitirá a SAPI acceder a las funcionalidades de nuestro motor a través del DDI.

En el presente proyecto fin de carrera nos centraremos en el segundo caso. Incluiremos el soporte de un motor TTS ya desarrollado, junto con la distribución libre de SAPI Microsoft incluye tres motores TTS en ingles, para demostrar como se haría uso de las funcionalidades que nos proporciona SAPI. Para ello modificaremos las clases *CWave* y *CWaveSink*, gracias a

[1] Para más detalles sobre el objeto consúltese la página web http://msdn.microsoft.com/library/en-us/wcesap/htm/cereflSpRecognier.asp.
[2] Consúltese la página web http://msdn.microsoft.com/library/en-us/wcesapi/cereflSpRecoResult.asp para más detalles.
[3] Más detalles en la página web http://msdn.microsoft.com/library/en-us/wcesap/htm/cereflSpVoice.asp.
[4] Consúltese la página http://msdn.microsoft.com/library/en-us/wcesap/htm/cereflSpSREngine.asp para una información más detallada del objeto.
[5] Información más detallada puede consultarse en la página web http://msdn.microsoft.com/library/en-us/wcesap/htm/cereflSpTTSEngine.asp.

la modularidad que se ha intentado mantener durante todo el desarrollo del sistema los cambios necesarios para modificar el sistema de reproducción de los mensajes quedan restringidos a estas dos clases, para incluir el soporte necesario para el motor TTS.

9.1 CWave

Esta clase reduce y modifica el número de métodos que implementa, modificando del mismo modo las variables miembro de la misma. El soporte para el TTS requiere que la clase implemente un objeto del tipo ISpVoice, para poder controlar y acceder a las funciones del motor, y otro del tipo ISpMMSysAudio[1], para dirigir la salida de audio generada por el motor hacia el teléfono. Del mismo modo los métodos que implementará para proporcionar el soporte necesario para la reproducción de texto a través del teléfono quedan reducidos a los que a continuación se citan:

- **bool Load(UINT nWaveOut):** Es la función encargada de inicializar el objeto para la reproducción del texto. La función se encarga de inicializar los objetos COM de SAPI necesarios, así como de configurarlos adecuadamente, tal y como se muestra en la siguiente ilustración.

[1] La página web http://msdn.microsoft.com/library/en-us/wcesap/htm/cereflSpMMSysAudio.asp contiene información más detallada sobre el objeto.

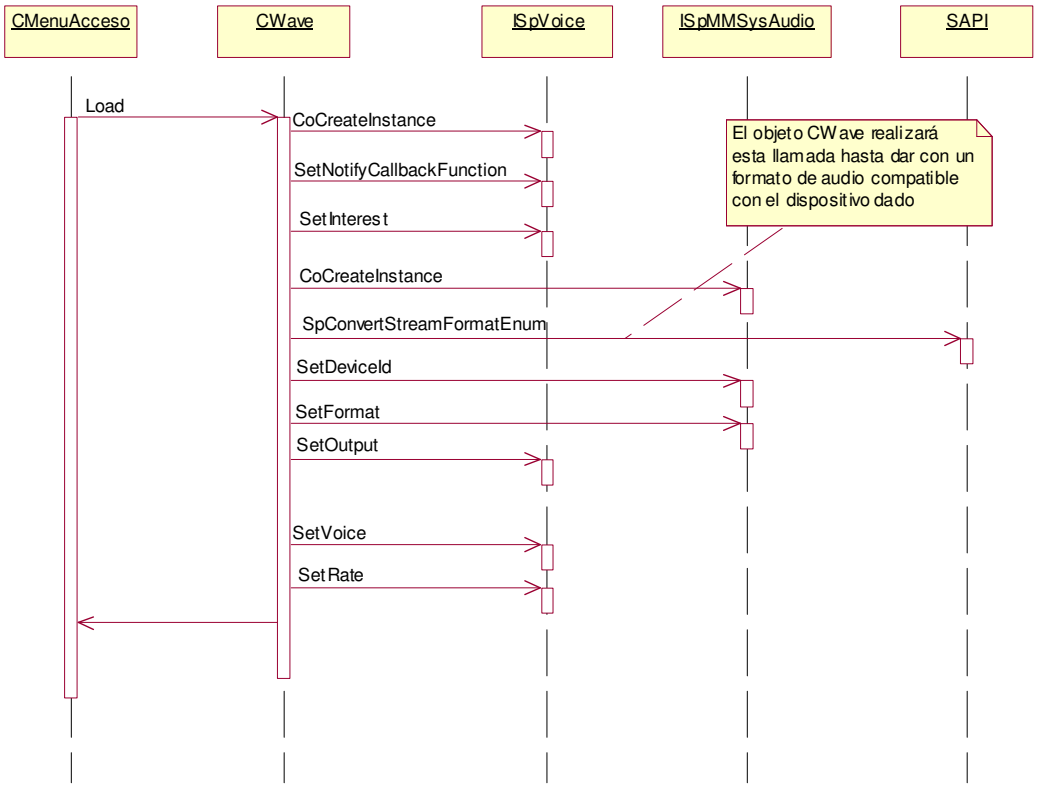

Ilustración 9-2

Como puede apreciarse en la ilustración, el primer paso necesario para poder hacer uso de los recursos del motor TTS es crear el objeto del tipo ISpVoice mediante el comando CoCreateInstance dando como identificador de clase el valor CLSID_SpVoice. Posteriormente se establecerá la función de callback que se desea que ejecute para la notificación de eventos a la aplicación. El objeto ISpVoice, al igual que ocurre con TAPI, genera eventos asociados a los cambios en el estado del objeto. Dichos eventos deben ser capturados por la aplicación que haga uso de dicho objeto con el objetivo de poder responder adecuadamente a esos cambios. El objeto ISpVoice admite diversos mecanismos para la notificación de eventos a la aplicación implementados a través el objeto ISpNotifySource[1] del que hereda, tales como un objeto del tipo ISpNotifySink[2] implementado por la

[1] Página web del objeto para más información sobre el mismo http://msdn.microsoft.com/library/en-us/wcesap/htm/cerefISpNotifySource.asp.
[2] Más detalles disponibles en la página web http://msdn.microsoft.com/library/en-us/wcesap/htm/cerefISpNotifySink.asp.

aplicación, un mensaje windows[1] o una función de callback. Para mantener la coherencia con el sistema anterior utilizaremos la función de callback, ya que es el sistema que introduce menos modificaciones. Así llamaremos al método SetNotifyCallbackFunction del objeto indicándole la función de callback así como los datos que serán dados como parámetros por SAPI cuando este ejecute la función. Dicha función de callback se ajusta a la siguiente definición:

```
void __stdcall SapiCallBack(WPARAM wParam, LPARAM lParam);
```

Donde los valores dados en los parámetros vienen especificados por los establecidos a la hora de llamar al método SetNotifyCallbackFunction, siendo el primero de ellos una referencia al objeto CWaveSink al que se reenviará el evento para su tratamiento. Como último paso en la configuración de los eventos que SAPI notificará a la aplicación se establece aquellos que deseamos que sean notificados a la aplicación. Para ello se emplea el método SetInterest[2] del objeto indicándole las listas de aquellos que deseamos que sean notificados y de aquellos que deseamos que SAPI almacene internamente, los cuales deben ser una parte de los anteriores. En nuestro caso solo estamos interesados en el evento de finalización de la conversión de texto a voz, así que ambas listas se reducen exclusivamente a este evento, identificado mediante la constante SPEI_END_INPUT_STREAM.

El paso siguiente en la inicialización del motor TTS para su utilización es el establecimiento del dispositivo de audio de salida que se empleará para la reproducción del texto, cuyo identificador ha sido dado como parámetro de la función, y el formato de la señal de audio que se generará. Se crea el objeto del tipo ISpMMSysAudio para redirigir la salida de SAPI hacia el MODEM. Al igual que

[1] La programación en entorno Windows se basa en la transferencia de distintos mensajes, cada uno con su propio significado y datos, entre los distintos objetos de la aplicación. Este mecanismo se basa en este principio, al evento de SAPI se le asignaría un mensaje que sería enviado al objeto de la aplicación que se le indique, el cual lo tratará.

[2] Dicho método posee dos parámetros que son un conjunto de variables del tipo SPEVENTENUM. Para más detalles sobre el método y el tipo de dato y sus posibles valores consúltense las siguientes página web: y http://msdn.microsoft.com/library/en-us/wcesap/htm/cerefISpNotifySourceSetInterest.asp y http://msdn.microsoft.com/library/en-us/wcesap/htm/cerefSPEVENTENUM.asp.

en el caso del objeto ISpVoice se realiza mediante el método CoCreateInstance, pero esta vez empleando el identificador de clase CLSID_SpMMAudioOut. En este paso hay que tener en cuenta que el formato que se seleccione debe ser compatible con el dispositivo que se vaya a emplear. Para evitar posibles fallos, en nuestro caso se ha optado por enumerar todos los formatos disponibles en SAPI, variables del tipo SPSTREAMFORMAT[1], y seleccionar el primero que resulte compatible con el dispositivo, para lo cual se convierte al formato wave mediante la función SpConvertStreamFormatEnum de SAPI y se comprueba mediante la función waveOutOpen. Encontrado el formato compatible se enlaza el objeto ISpMMSysAudio creado con el dispositivo de audio dado, se establece el formato compatible y se enlaza el objeto ISpMMSysAudio con el objeto de control del motor TTS mediante el método SetOutput.

El siguiente paso en la inicialización del motor TTS consiste en la selección de la *voz*, es decir, del motor propiamente dicho. Los motores registrados en el ordenador se recogen en el registro de Windows bajo la clave HKEY_LOCAL_MACHINE\SOFTWARE\MICROSOFT\SPEECH\VOICES\TOKENS y pueden ser enumerados empleando la función auxiliar de SAPI SpEnumTokens[2] o seleccionados específicamente mediante la función SpGetTokenFromId[3], y almacenados en objetos del tipo ISpObjectToken[4]. En nuestro caso seleccionaremos uno de los que se instalan junto a SAPI, para lo cual seleccionaremos el que se establece por defecto, para lo cual emplearemos el método SpGetDefaultTokenFromCategoryId[5] indicándole el tipo de objeto a seleccionar mediante el identificador de categoría SPCAT_VOICES y que nos devolverá el objeto ISpObjectToken de la voz seleccionada. Una vez obtenida la *voz* se establece en el objeto ISpVoice mediante el método SetVoice, cuyo único

[1] Véase la página y http://msdn.microsoft.com/library/en-us/wcesap/htm/cerefSPSTREAMFORMAT.asp para un listado completo de los formatos soportados por SAPI.
[2] Para más detalles consúltese la página web http://msdn.microsoft.com/library/en-us/wcesap/htm/cerefSpEnumTokens.asp.
[3] Más información sobre la función esta disponible en la página web http://msdn.microsoft.com/library/en-us/wcesap/htm/cerefSpGetTokenFromId.asp.
[4] Dicho tipo de objeto es empleado por SAPI para manejar los accesos a los distintos tipos de recursos registrados y disponibles en la máquina, tales como voces, motores SR o dispositivos de audio. Para más detalles consúltese la página web: http://msdn.microsoft.com/library/en-us/wcesap/htm/cerefISpObjectToken.asp.
[5] Consúltese la página web http://msdn.microsoft.com/library/en-us/wcesap/htm/cerefSpGetDefaultTokenFromCategoryId.asp para obtener más detalles sobre la función y sus parámetros.

parámetro es una referencia al objeto ISpObjectToken que representa la voz a establecer.

Por último, y como finalización de la configuración del motor TTS, estableceremos la velocidad de dicción del mismo, estableciéndola un poco por debajo del valor natural, 0, con el fin de ralentizar la dicción lo que favorecerá la comunicación y el correcto entendimiento de los mensajes por parte del usuario.

- **bool Play(char* szText):** Esta función es la destinada a reproducir el texto dado como parámetro a través de SAPI. Para ello se basa en dos métodos del objeto ISpVoice: Speak[1], que "leerá" el texto dado como parámetro, y Resume[2], que pone el dispositivo de audio de salida en estado activo. Tras ello el motor TTS generará la señal de voz basándose en el texto dado y dirigirá dicha señal hacia el dispositivo de salida establecido.

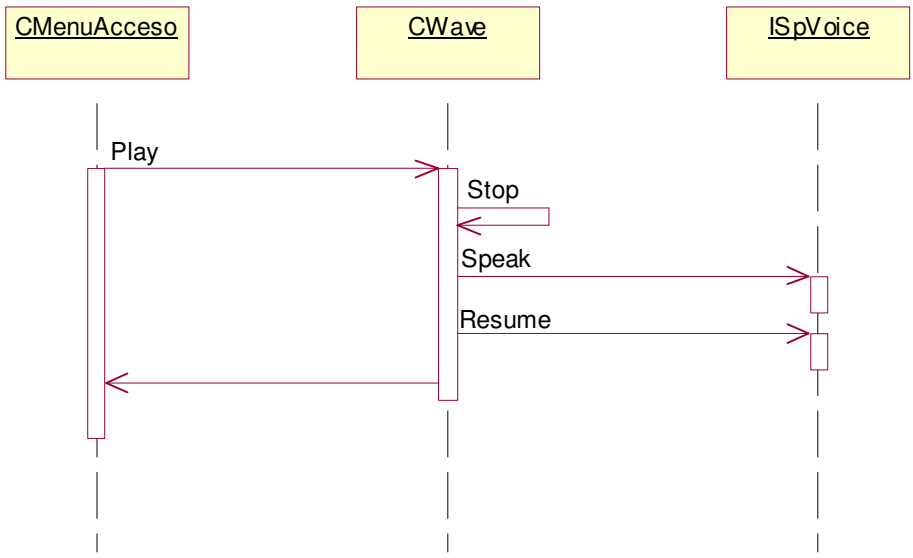

Ilustración 9-3

[1] Para más detalles sobre el método y sus parámetros consúltese la página web http://msdn.microsoft.com/library/en-us/wcesap/htm/cereflSpVoiceSpeak.asp.
[2] Información más detallada sobre el método esta disponible en la página web http://msdn.microsoft.com/library/en-us/wcesap/htm/cereflSpVoiceResume.asp.

- **bool Stop():** La finalidad de esta función es la de detener la reproducción del texto, para lo cual purga el objeto ISpVoice llamando al método Speak pero dando como segundo parámetro la constante SPF_PURGEBEFORESPEAK. Esto provoca que el objeto ISpVoice descargue todas las peticiones de reproducción que tuviese pendientes.

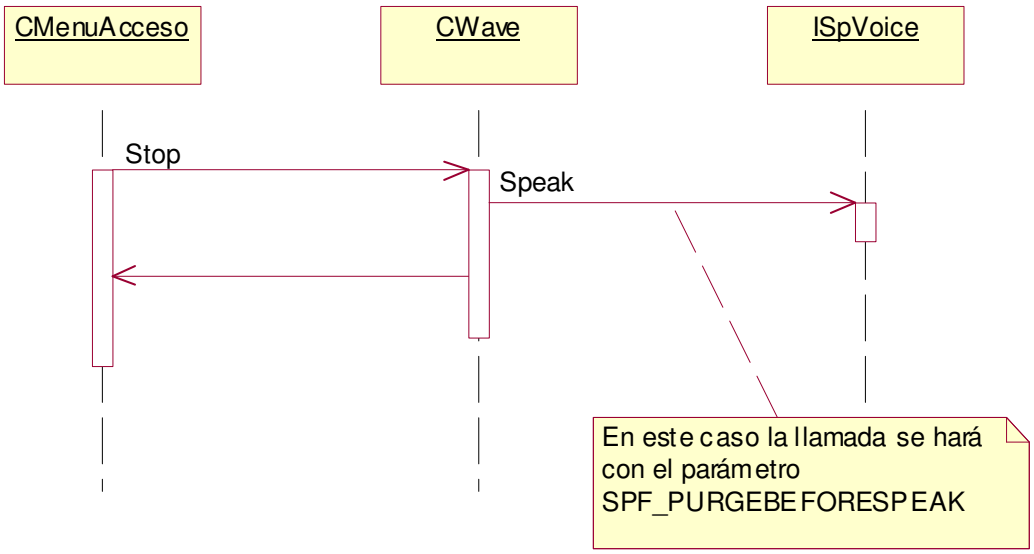

Ilustración 9-4

- **bool Close():** Esta función finalizar el objeto CWave liberando los objetos ISpVoice e ISpMMSysAudio creados. Para ello emplea el método Release sobre cada uno de ellos, lo que desliga el objeto CWave de los ya mencionados, permitiendo que se descarguen y cierren adecuadamente.

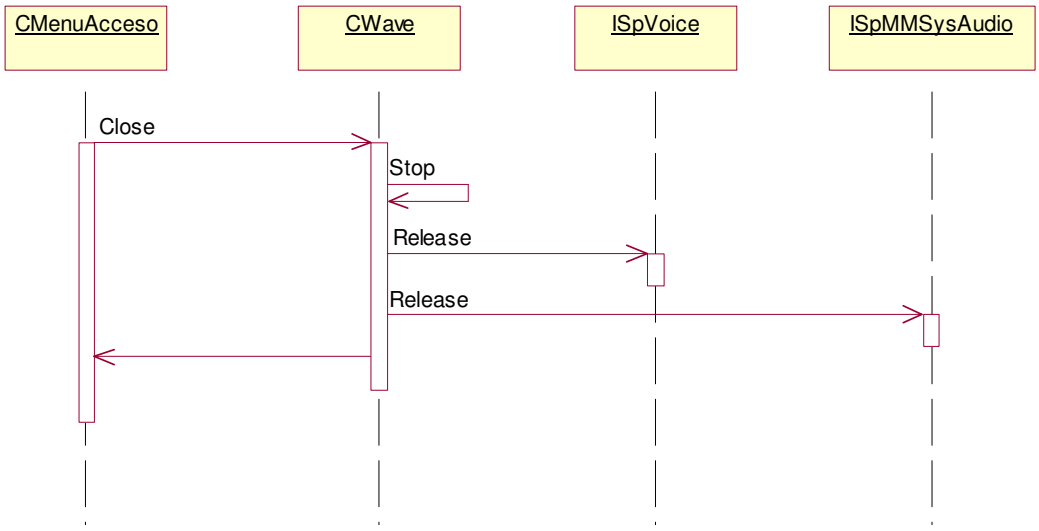

Ilustración 9-5

10 Anexo 3: SMS

La solución planteada en el presente proyecto fin de carrera se basa en el empleo de un MODEM analógico y de la línea telefónica fija analógica como mecanismos para la comunicación entre el usuario y el sistema de control domótico, sin embargo, como ya se ha comentado, existen otras posibles soluciones. De todas ellas la que ofrece mayor atractivo, teniendo en cuenta la expansión que han tenido los sistemas de telefonía móvil, es la basada en el empleo de sistema de mensajes SMS para la comunicación entre el usuario y el sistema de control domótico.

Dicho sistema estaría basado en un conjunto de mensajes predefinidos cuyo formato y contenido estarían ajustados a la funcionalidad destinada a los mismos. Así se establecería un formato, o formatos, para el envío de comandos de actuación sobre el sistema, del tipo, por ejemplo, nombre del sistema seguido del valor de actuación. Del mismo modo existiría otro juego de mensajes para la petición de información sobre el sistema, así como mensajes de información y aviso que el sistema enviaría al usuario.

Para la realización de esta opción podría recurrirse al uso de un MODEM GSM como pasarela entre el sistema de control domótico, supuesto que este se monte sobre un ordandor tipo PC como en el presente proyecto fin de carrera, y la red de telefoní móvil. Dichos aparatos, como por ejemplo el modelo G2 de Wavecom, están disponibles en el mercado y su funcionamiento es similar a los actuales teléfonos móviles. Disponen de una ranura para la introducción de la tarjeta SIM que nos proporcionará el acceso a la red del operador que seleccionemos y las funciones que ofrecen son casi idénticas a las de cualquier teléfono GSM actual. Además su conexión al PC se realiza mediante el puerto serie, aunque algunos más modernos emplean los puertos USB para comunicarse con el PC. Por lo tanto su empleo y conexión resultan sencillas y no suponen una complejidad que pueda suponer un lastre para su uso.

En cuanto a la programación de las funciones que el MODEM GSM debe ofrecernos para el desarrollo del interfaz entre el usuario y el sistema de control domótico, cabe destacar que

este tipo de MODEMs soportan un juego de comandos muy similar al que soportan los MODEMs analógicos. Este juego de comandos, comandos AT, se encuentra definido con el ETSI en su especificación ETSI GSM 07.07 y esta disponible a través de la página web de dicho organismo[1], aunque muchos de los fabricantes de este tipo de dispositivos amplían dicha especificación con sus propios comandos, en cuyo caso dichas especificaciones estarán disponibles a través de los respectivos fabricantes. De forma general, el juego de comandos AT para GSM permite acceder a la totalidad de las funciones del MODEM GSM y de los servicios de la red de telefonía móvil, pudiéndose acceder a funciones tales como el listín telefónico de la tarjeta SIM o al envío y recepción de mensajes SMS.

Por último es de destacar la posibilidad de emplear TAPI en conjunción con el MODEM GSM. Como ya se ha comentado TAPI se basa en un conjunto de dos interfaces, uno hacia la aplicación que quiera hacer uso de las funciones de TAPI, y otro hacia el dispositivo, el TSPI. De este modo si el fabricante del MODEM GSM ha desarrollado su parte correspondiente al TSPI, TAPI podrá hacer uso del MODEM GSM para ofrecer sus funcionalidades.

[1] http://www.etsi.org

11 Anexo 4: Guía del Usuario

11.1 Introducción

Como parte del presente Proyecto Fin de Carrera se ha realizado este manual de usuario con el fin de proporcionar un método rápido de consulta del proceso de instalación, uso y programación del Módulo del Interfaz Telefónico. Además junto con la documentación que acompaña al presente Proyecto Fin de Carrera se incluye una aplicación de demostración del funcionamiento y uso del Módulo del Interfaz Telefónico, como parte complementaria a este manual.

La instalación y uso del Módulo del Interfaz Telefónico se divide en dos campos: uno enfocado a la introducción de las librerías necesarias dentro del proyecto de la aplicación de control domótico, a partir de ahora aplicación base, de forma que los recursos de los que dispone este accesibles desde y para la aplicación base; y otro destinado a la programación y uso de las funciones que ofrece el Módulo del Interfaz Telefónico.

11.2 Uso del Módulo del Interfaz Telefónico

En este apartado se explicarán los pasos necesarios para la introducción del Módulo del Interfaz Telefónico en la aplicación base. Dicho procedimiento permitirá generar la aplicación base con el soporte que proporciona el Módulo del Interfaz Telefónico de forma que podrá acceder a las funciones que este implementa.

Como paso previo a la introducción cabe señalar que el Módulo se ha desarrollado para su empleo dentro del entorno de programación Visual C++, de forma que supondremos que la aplicación base esta generada en dicho entorno.

La introducción del Módulo del Interfaz Telefónico se basa en un conjunto de ficheros de definiciones y de librería que deben ser incluidos dentro del proyecto de la aplicación base. Para ello se crea un directorio dentro de la carpeta del proyecto de la aplicación base llamado

Lib, dentro del cual se copiaran los siguientes ficheros, disponibles en el directorio Includes del CD que acompaña al presente Proyecto Fin de Carrera:

AddressCaps.h	Cabecera de la clase CAddressCaps. Esta clase ofrece funciones para el acceso a las capacidades de una dirección. Ofrece funciones para el acceso a los datos de las capacidades.
AddresStatus.h	Cabecera de la clase CAddressStatus. Esta clase ofrece funciones para el acceso al estado de una dirección. Ofrece funciones para el acceso a los datos del estado.
Aviso.h	Cabecera para la clase CAviso. Clase de control de un aviso.
Call.h	Cabecera de la clase CCall. Clase de manejo de las llamadas. Ofrece funciones para el manejo de dichas llamadas.
CallInfo.h	Cabecera de la clase CCallInfo. Clase de acceso a los datos de información de la llamada.
CallStatus.h	Cabecera para la clase CCallStatus. Clase de acceso a los datos del estado de una llamada.
ControlAverias.h	Cabecera para la clase CControlAverias. Clase de control de las notificaciones de avisos.
ControlLine.h	Cabecera de la clase CControlLine. Clase de control de la línea. Ofrece funciones para la inicialización de TAPI y para el manejo de la línea.
ControlMenu.h	Cabecera para la clase CControlMenu. Clase de control del menú telefónico.
CountryList.h	Cabecera de la clase CCountryList. Esta clase ofrece funciones para la recuperación de la lista de países definida por TAPI así como de los datos de los distintos países de la lista.
DeviceID.h	Cabecera de la clase CDeviceID. Esta clase ofrece funciones para la recuperación de los identificadores de dispositivos asociados a una línea, llamada o dirección de línea
Line.h	Cabecera de la clase CLine. Clase de control de la línea. Ofrece funciones para el manejo de la línea.
LineApp.h	Cabecera para la clase CLineApp. Clase de inicialización de TAPI.
LineDevCaps.h	Cabecera para la clase CLineDevCaps. Ofrece funciones para el acceso a las capacidades de un LineDevice una vez iniciado TAPI.
LineSink.h	Cabecera de la clase CLineSink. Clase base para la recepción de eventos de la línea.
MenuAcceso.h	Cabecera de la clase CMenuAcceso. Clase de gestión del menú de control de acceso.
MenuAccion.h	Cabecera de la clase CMenuAccion. Clase de gestión del menú de actuación sobre el sistema.
MenuInformacion.h	Cabecera de la clase CMenuInformacion. Clase de gestión del menú de petición de información del sistema.
MenuInicio.h	Cabecera de la clase CMenuInicio. Clase de gestión del menú inicial.
MenuOpciones.h	Cabecera de la clase CMenuOpciones. Clase de gestión del menú de opciones de actuación o petición de información.
MenuSink.h	Cabecera de la clase CMenuSink.

	Clase base para la recepción de eventos del menú.
MenuTelfCtrl.h	Fichero de includes del Módulo del Interfaz Telefónico.
OpcionMenu.h	Cabecera de la clase COpcionMenu. Clase base de los menús de la aplicación.
Timer.h	Cabecera de la clase CTimer. Clase de control del timer de la llamada. Ofrece funciones para el manejo del timer.
TimerSink.h	Cabecera de la clase CTimerSink. Clase base para la recepción de eventos del timer.
TranslateCaps.h	Cabecera de la clase CTranslateCaps. Clase de acceso a los datos de las localización y tarjetas.
TranslateOutput.h	Cabecera de la clase CTranslateOutput. Esta clase ofrece funciones para las conversiones entre los distintos formatos de representación de los números telefónicos.
Wave.h	Cabecera de la clase CWave. Clase de carga y reproducción de los ficheros de sonido de los mensajes.
WaveSink.h	Cabecera de la clase CWaveSink. Clase base para la recepción de eventos del wave.
MenuTelfCtrl.lib	Libreria del Módulo del Interfaz Telefónico.

De todos estos ficheros, los más destacados son MenuTelfCtrl.h y MenuTelfCtrl.lib. Estos dos ficheros forman en núcleo del Módulo del Interfaz Telefónico que debe ser incluidos en el proyecto de la aplicación base. Para ellos deberán realizarse los siguientes procedimientos:

1. **Incluir la librería del Módulo del Interfaz Telefónico.** Para ello se seleccionará la opción *Settings* dentro del menú *Project*. Una vez abierto el dialogo de configuración se seleccionará la pestaña *Link* eligiéndose la categoría *General* donde se incluirá en el cuadro de texto *Object/library modules* el nombre de la librería del Módulo del Interfaz Telefónico, MenuTelfCtrl.lib. Seguidamente se seleccionará la categaría *Input* donde en *Additional library path* se incluirá el directorio .\Lib. Por último se selccionará la pestaña *C/C++* eligiéndose la categaría *Preprocessor* donde en el campo *Additinal include directories* se incluirá el directorio .\Lib.

2. **Copia del Módulo del Interfaz Telefónico y del Módulo de Sonidos.** Como siguiente paso en la inclusión del Módulo del Interfaz se deben copiar los ficheros DLL de los módulos mencionados en el directorio donde se ejecutará la aplicación base. Dichos ficheros, MenuTelfCtrl.dll y WaveLib.dll, se hallan en el directorio Includes del CD que acompaña al presente Proyecto Fin de Carrera.

3. **Introducción de las definiciones del Módulo del Interfaz Telefónico.** Finalmente se deberá incluir en el proyecto de la aplicación base el fichero de definiciones del Módulo del Interfaz Telefónico, el fichero MenuTelfCtrl.h. Para ello se incluirá la línea #include "MenuTelfCtrl.h" en uno de los ficheros de cabeceras principales de la aplicación, en el caso de la aplicación de demostración que acompaña al presente Proyecto Fin de Carrera se ha incluido en el fichero Stdafx.h.

Una vez realizados los procedimientos anteriormente comentados, la aplicación base podrá acceder a las clases y procedimientos definidos en el Módulo del Interfaz Telefónico. De este modo la aplicación base podrá instanciar objetos de las clases definidas en el interfaz y ejecutar los métodos de las mismas.

11.3 Programación del Módulo del Interfaz Telefónico

Una vez habilitado el Módulo del Interfaz Telefónico para su uso por parte de la aplicación base esta deberá cumplir ciertos requisitos en su desarrollo y programación para poder hacer uso de toda la funcionalidad que el desarrolla el Módulo del Interfaz Telefónico. Así se deberá tener en cuenta una serie de pequeñas restricciones, así como unos procedimientos a la hora de realizar ciertas operaciones. Dichas limitaciones se comentan a continuación.

11.3.1 Herencia de la clase CMenuSink

La principal restricción a la hora de desarrollar la aplicación base radica en la necesidad de heredar alguna de sus clases de la clase CMenuSink implementada por el Módulo del Interfaz Telefónico. Dicha clase da soporte al mecanismo de comunicaciones entre la aplicación base y las restantes clases que formar el Módulo. Gracias a esta clase el Módulo es capaz de comunicar a la aplicación base aquellos eventos que ocurran dentro del interfaz y que sean de importancia para el correcto funcionamiento del sistema. De este modo se hace preciso que la

aplicación base implemente una clase de este tipo, siendo necesaria la herencia pues se trata de una clase virtual[1], para luego registrarla en el Módulo de forma que este sea capaz de acceder a ella para comunicarle los eventos.

En la aplicación de pruebas que acompaña al presente Proyecto Fin de Carrera dicha herencia se ha realizado en la clase CMainFrame por tratarse del objeto principal de la aplicación. Dicha clase hereda públicamente de la clase CMenuSink del interfaz, implementando los métodos definidos por esta última, de forma que se les ha proporcionado una funcionalidad de acuerdo a los requisitos propios de la aplicación de pruebas. De este modo el Módulo del Interfaz Telefónico llamará a uno de los métodos, dependiendo del evento que quiera comunicar, por lo que como respuesta a dicho evento la aplicación de pruebas ejecutará un determinado procedimiento ajustado a sus propias necesidades.

11.3.2 Instanciación de la clase CMenuControl

Como segunda restricción en el desarrollo de la aplicación base se encuentra la necesidad de instanciar un objeto de la clase CMenuControl, es decir, la aplicación base debe crear un objeto de dicha clase para poder acceder a las funciones proporcionadas por el Módulo del Interfaz Telefónico.

La clase CMenuControl actúa como núcleo del Módulo, siendo la responsable de crear y mantener las restantes clases que forman el interfaz y que le proporcionan a este toda su funcionalidad. Además proporciona los mecanismos necesarios para la configuración del Módulo por parte de la aplicación base y para la comunicación de los avisos/emergencias. Es por todo ello que se hace del todo imprescindible que la aplicación base cree un objeto de esta clase al que se puede acceder.

En la aplicación de pruebas que acompaña al presente Proyecto Fin de Carrera el objeto de la clase CMenuControl se ha localizado nuevamente en la clase CmainFrame, por ser esta

[1] Una clase virtual no implementa los métodos que tiene definidos por lo que por si sola no realiza función alguna. Sirven para definir interfaces que luego serán implementados por otras clases mediante la herencia.

la clase principal de la aplicación. De este modo se podrá acceder a los métodos y funciones del Módulo del Interfaz Telefónico a partir de dicho objeto, estando además disponible para las restantes clases de la aplicación de pruebas que requieran del objeto.

11.3.3 Inicialización del Interfaz

El proceso de inicialización del Módulo del Interfaz Telefónico se realiza siguiendo en procedimiento que se muestra en la siguiente ilustración.

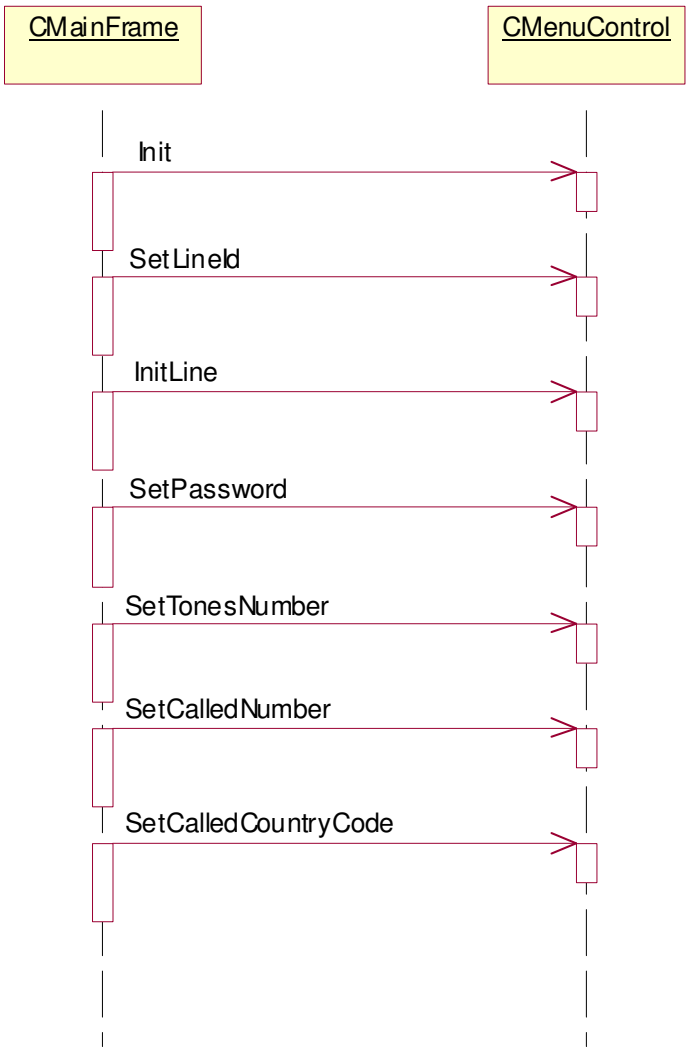

Ilustración 11-1

Como puede apreciarse el proceso de inicialización se realiza siguiendo una serie de pasos destinados a configurar los distintos parámetros que actúan sobre el funcionamiento del interfaz, clave de acceso, número de tonos de espera, número telefónico al que se llamara en caso de aviso/emergencia y el identificador del país de dicho número telefónico.

Del mismo modo se inicializan los objetos de control de la línea telefónica, comandos *Init* e *InitLine*, este último tras haber establecido el identificador de la línea que se desea abrir y a la que se conectará el Módulo del Interfaz Telefónico a través de TAPI. El primero de estos procedimientos establece en el Módulo, y este a su vez en TAPI, el nombre de la aplicación que lo crea así como el manejador de la instancia de la misma, datos que usará TAPI para enlazarse con la aplicación.

11.3.4 Finalización del Interfaz

Al igual que la inicialización, la finalización del Módulo del Interfaz Telefónico se realiza siguiendo un determinado procedimiento destinado a liberar los recursos ocupados por el Módulo así como a cerrar la línea que pudiere estar abierta. Dicho procedimiento se muestra a continuación.

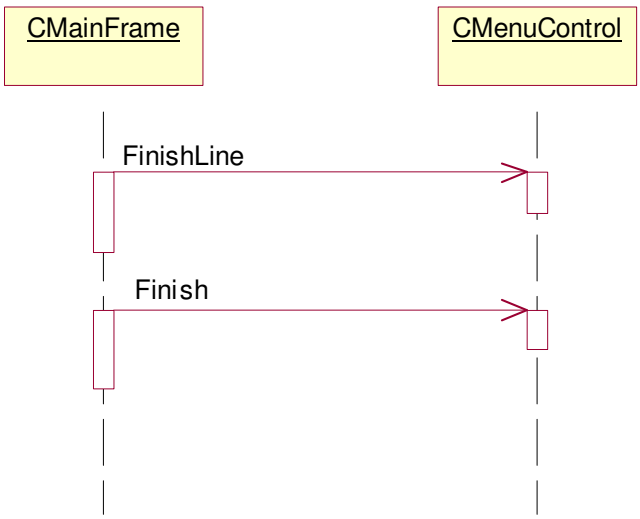

Ilustración 11-2

Como puede apreciarse el procedimiento esta basado en únicamente en dos métodos, *FinishLine*, destinado a cerrar la línea que en el momento de cerrar el Módulo pudiera estar en uso, y *Finish*, destinado a liberar y descargar todos aquellos recursos que el Módulo haya creado u ocupado .

11.3.5 Comunicación de Avisos/Emergencias

En este caso el procedimiento que se debe seguir para la comunicación de avisos/emergencias al Módulo del Interfaz Telefónico desde la aplicación base es muy sencillo. Se basa en un único método de la clase CMenuControl que la aplicación base debe llamar cada vez que desee comunicar un nuevo aviso/emergencia al Módulo. Dicho procedimiento puede observarse en la siguiente ilustración.

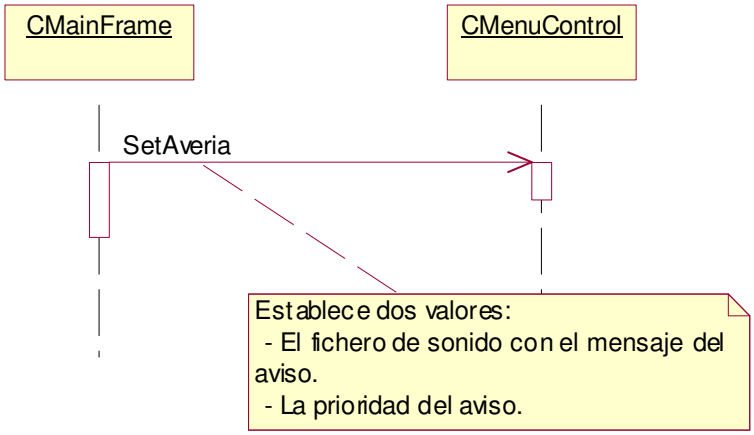

Ilustración 11-3

Dicho método crea un nuevo aviso en el Módulo del Interfaz Telefónico, asignándole la prioridad y el fichero de sonido establecidos en la llamada al método.

11.3.6 Configuración del los Parámetros del Interfaz

Los parámetros del interfaz, tales como la clave o el identificador de la línea, pueden ser modificados en cualquier momentos por la aplicación base. Para ello basta con que la aplicación base llame al correspondiente método del la clase CMenuControl para que el cambio

se haga eficaz. La única salvedad se localiza en el cambio del identificador de la línea, ya que en dicho caso, y antes de realizar la llamada al método *SetLineId*, la aplicación base debe cerrar la línea abierta, en caso de que la hubiera. Una vez cerrada se establecería el nuevo identificador y se volvería a abrir la nueva línea. Este último proceso se detalla en el siguiente diagrama de flujo.

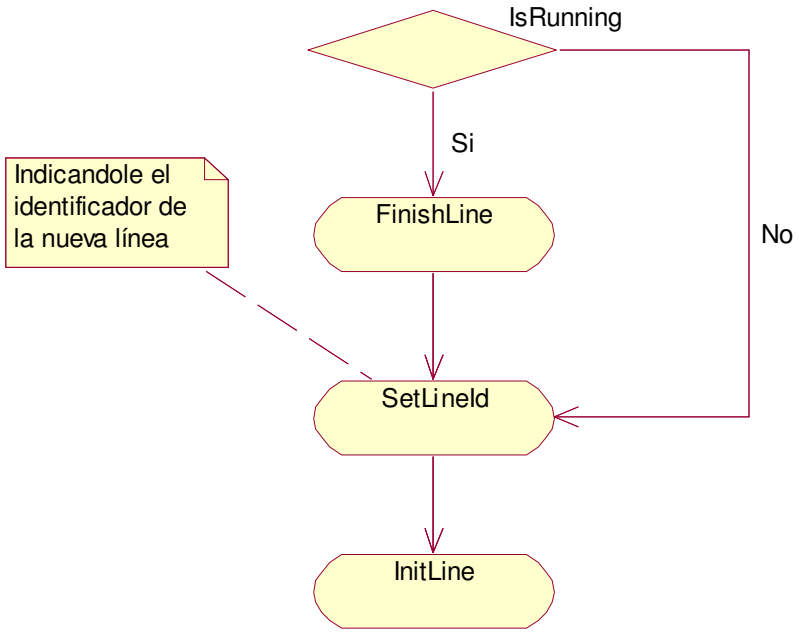

Ilustración 11-4

11.4 Voz y SAPI

En caso de instalarse la versión del Módulo del Interfaz Telefónico los pasos a seguir para la integración del mismo con la aplicación base son muy similares a los ya comentados, solo que en algunos puntos se requiere de alguna operación adicional.

11.4.1 Instalación de SAPI

Los recursos disponibles de SAPI no están disponibles en las instalaciones del sistema operativo Windows ni en las instalaciones del entorno de desarrollo Visual C++. Por lo tanto su

uso requiere de la instalación previa de esta herramienta, para lo cual habrá que instalar el *Microsoft Speech SDK 5.1*, disponible para su descarga desde la pagina web de Microsoft[1] o el directorio *Speech SDK* del CD que acompaña al presente Proyecto Fin de Carrera. Para su instalación se ejecuta el programa *Setup* del SDK y se siguen las instrucciones que el programa de instalación nos vaya dando.

11.4.2 Uso del Módulo del Interfaz Telefónico

En este punto los pasos a seguir para habilitar el uso del los recursos disponibles en el Módulo del Interfaz Telefónico por parte de la aplicación base son los mismos que en el caso de emplearse la versión sin soporte de Voz, salvo las siguiente observación:

1. En los *Settings* del proyecto deben realizarse dos modificaciones. Seleccionando la etiqueta *C/C++* y dentro de ella la categoría *Preprocessor*, en el campo *Aditional Include Directories* hay que incluir el directorio de definiciones del Speech SDK, habitualmente se trata del directorio *Includes* dentro del directorio donde se instaló[2]. Del mismo modo, seleccionando la etiqueta *Link* y dentro de la categoría *Input*, en el campo *Additional library path* hay que incluir el directorio de librerías del Speeck SDK, habitualmente el directorio lib\i386 dentro del directorio donde se instaló[3].

11.4.3 Programación del Módulo del Interfaz Telefónico

En este caso las modificaciones tampoco resultan significativas ya que ninguno de los procedimientos anteriormente comentados se ve afectado, con la única salvedad de la información que se establece en la notificación de avisos/emergencias desde la aplicación base al Módulo del Interfaz Telefónico. Al utilizar el motor TTS accesible desde SAPI para la conversión de texto a voz en la reproducción de los mensajes, los parámetros de la llamada a

[1] http://www.microsoft.com/downloads/details.aspx?FamilyID=5e86ec97-40a7-453f-b0ee-6583171b4530&DisplayLang=en
[2] C:\Archivos de programa\Microsoft Speech SDK 5.1\Include en la instalación por defecto.
[3] C:\Archivos de programa\Microsoft Speech SDK 5.1\lib\i386 en la instalación por defecto.

SetAveria de la clase CMenuControl cambian, pasando a ser el texto del menaje que se reproducirá con el aviso/emergencia y la prioridad del mismo.

12 Bibliografía y referencias

1. Windows Telephony Programming, Chris Sells, Assison-Wesley Pub Co, 1999.
2. Windows Telephony: A Practical Guide to Designins, Using and Developing Telephony Applications on All Windows Operating Systems, Jeffrey R. Shapiro, CMP Books, 1996.
3. Computer Telephony Integration, 2º Edición, William A. y Jr. Yarberry, Aurbach Pub, 2002.
4. Computer Telephony Demystified, Michael Bayer, McGraw-Hill Professional, 2000.
5. Computer Telephony Encyclopedia, Richard Grigonis, CMP Books, 2000.
6. Mapi, Sapi and Tapi: Developer's Guide, Michael C. Amundsen, Sams, 1996.
7. PC-Telephony: The Complete Guide to Designing, Building and Programming Systems Using Dialogic and Related Hardware, Bob Edgar, CMP Books, 1998.
8. http://www.microsoft.com/windows2000/docs/tapi30.doc.
9. http://msdn.microsoft.com/library/en-us/tapi/tapi2/tapi_2_2_reference.asp
10. http://msdn.microsoft.com/library/en-us/tapi/tapi3/tapi_3_1_start_page.asp
11. http://allen-martin-inc.com/amtapitelephony.asp
12. http://www.vbip.com/miscellaneous/tapi-01.asp
13. http://project.uet.itgo.com/tapi.htm
14. http://www.microsoft.com/windows2000/techinfo/howitworks/communications/telephony/tapiincoming.asp
15. http://www.cconvergence.com/article/CTM20000512s0020
16. http://msdn.microsoft.com/library/en-us/wcesapi/htm/ceoriSpeechAPISAPIVersion50
17. http://msdn.microsoft.com/library/en-us/wcsapi/htm/ceconsapi50overview.asp
18. http://www.codeguru.com/forum