UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO
FACULTAD DE INGENIERÍA

DESARROLLO DE UN SISTEMA DE MONITOREO DE SEÑALES DE AUDIO Y VIDEO PARA TELEVISORAS

Tesina

que para obtener el título de

Ingeniero en Computación

Realizado por ***SALVADOR BEDREDÍN MEDINA MAZA***

bajo la supervisión de ***GILBERTO MARRÓN VILLAFÁN***

Ciudad Universitaria 2012

In Memoriam

Salvador Medina Manríquez
Isabel Ramírez Otero
Abraham Montiel

Agradecimientos

Quiero agradecerle a mi mamá por propiciar un ambiente adecuado para mi formación personal y profesional. Le debo gran parte de lo que soy hoy en día y este trabajo es una respuesta al apoyo incondicional que me ha dado a través de mi vida.

A ti Julie por hacerme ver lo valioso que soy y demostrarme de lo que soy capaz viéndome como una figura a seguir. Semper frateri.

A ti Chente por apoyarme en la base de mi educación con incontables tardes de cultura y tareas a resolver y estribar mi manera de percibir el conocimiento durante la inmortalidad del cangrejo.

A ti Mariana, por aceptar ser mi adorable esposa y apoyar mis ideas en todo momento, por estar ahí en esos momentos de desesperanza y nunca dar por vencida la idea de titularme para lograr mis anhelos y deseos en el ámbito profesional y académico. Te amo.

También quiero agradecer a mis suegros por apoyarme en los últimos años, favoreciendo un ambiente con menos preocupaciones, mediante el cuál pude enfocarme a realizar este escrito.

Gracias a la Universidad Nacional Autónoma de México por acobijarme durante mi formación como profesionista y otorgarme la oportunidad de vislumbrar un México distinto al que conocía anteriormente. Agradezco a la Facultad de Ingeniería por permitirme ser parte de esa gran familia académica, tan noble, donde nos inculcan ser mejores cada día para así poder formar un México mejor.

Dr. Jesús Savage, muchas gracias por permitirme ser uno de sus pupilos y partícipe de sus proyectos. Usted me introdujo en el desarrollo y la investigación de una manera formal, en la elaboración de proyectos relevantes y otórgame la oportunidad de trabajar en un equipo. Es un gran ejemplo a seguir y este escrito se lo debía desde hace muchos años.

También quiero dar las gracias a Fission Software y en especial a Artemio por invitarme a ser parte de un equipo de trabajo sin igual. Me han permitido crecer en el ámbito profesional confiando en mis habilidades para la elaboración de proyectos tan interesantes. Son una gran familia para mí.

Gracias a Abraham Monrroy, Abraham Segura, Carlos Alegría y Francisco Rodríguez, por guiarme en la elaboración de este escrito y retroalimentarme con sus comentarios durante las revisiones que hicieron. Porque a pesar de los años que tomó hacerlo, confiaron en que algún día lo terminaría. También le agradezco a Carlos Velasco la revisión final que hicimos de una manera bastante dinámica, como lo solíamos hacer durante nuestra educación universitaria. Lo logramos.

Y a ti, porque a pesar de que te encontré tarde, te encontré, para que seas mi inspiración a seguir, el pilar en mi mejora como ser humano. Ahí espérame por que en algún tiempo y espacio nos volveremos a reunir.

Índice

Índice de Figuras

Índice de Tablas

Introducción

El propósito de este reporte es describir mi trabajo, de una manera general, dentro de Fission Software de México S.A. de C.V.. La compañía principalmente me contrató con el fin de desarrollar un producto dedicado al monitoreo de señales de audio y video transmitidos por una televisora. Por lo tanto, trataré de explicar, por este medio, la manera en que diseñé y desarrollé un producto de software basado en hardware especializado, que se originó en un área de oportunidad dentro de las necesidades de trabajo de una estación de televisión.

La implementación de un testigo de señal de retorno, es decir, contar con una forma de grabar sin interrupción las señales de audio y video transmitidas por las televisoras, para ser revisadas posteriormente por un usuario de manera visual y auditiva.

Esta operación es importante para una televisora, por ejemplo, para obtener una forma de comprobar a sus clientes que sus comerciales se han transmitido como se estipuló en sus contratos. Asimismo, con esta herramienta pueden corroborar inmediatamente si existe algún fallo en la señal que se trasmitió al aire brindando una velocidad de respuesta mayor ante el error. Hoy en día, para solucionar este tipo de problemas, en diversas estaciones de televisión de México y Sudamérica, todavía utilizan DVR's o videocaseteras.

La desventaja de utilizar esos dispositivos es que los tiempos de captura son muy limitados, brindando en promedio una semana de grabación. Además estos aparatos se activan manualmente por medio de un operador. Otra desventaja que presentan estos dispositivos es que requieren de un mantenimiento constante para que funcionen correctamente, ya que su uso continuo degenera el equipo con el paso del tiempo.

Para resolver esta problemática, Fission Software buscó desarrollar un sistema capaz de grabar señales de audio y video en unidades de almacenamiento masivo y escalable, para tener el mayor tiempo de grabación posible. También escrutó que el sistema sea lo suficientemente autónomo para poder realizar un mantenimiento periódico de las unidades de almacenamiento utilizadas, obteniendo de esta forma el espacio libre suficiente para mantener la grabación activa en todo momento, mediante la eliminación de los archivos más viejos o que ya no sea necesaria su existencia.

En respuesta a esta necesidad, el producto que implementé se denomina Fission On-Air Monitor. Un sistema de monitoreo de señales de televisión basado en el levantamiento de requerimientos, realizado en base a las necesidades del mercado.

Capítulo 1 - Fission Software

Fission Software es una corporación, fundada en el año 2000 incorporada al estado de Connecticut, E.U.A., dedicada principalmente al desarrollo de soluciones para automatización de áreas de Control Maestro (Master Control) de estaciones de televisión.

En el año 2001, abrió una subsidiaria llamada Ground Zero Software de México, S.A. de C.V. (GZSM) con el propósito de penetrar en el mercado latinoamericano, teniendo un mejor alcance a clientes ubicados dentro Centro y Sudamérica, así como el área del Caribe.

En el año 2002, GZSM ganó la licitación para requipar todas las estaciones locales de Televisa, así como sus principales estaciones repetidoras de señal en el interior de la República Mexicana. Posteriormente, estaciones como Televisa Monterrey, Televisa Guadalajara, por mencionar algunas, fueron actualizadas con hardware y software de última generación en los años 2003 y 2004.

Al mismo tiempo, en el 2004, se crea Fission Software de México, subsidiaria donde comienza la labor de distribuir la solución en América Latina, mientras que el producto continúa consolidándose en el mercado doméstico con clientes como: Televisa Networks, Sky, Megacable, MVS, entre otras. Lugar donde he laborado desde el 2006 hasta la fecha.

Para el año 2005, Fission Asia Limited se constituye como mayorista para la región de Asia y el Pacífico, obteniendo de esta manera presencia en países como Corea, Taiwan, Australia, etc.

Después en el 2006 se abre comercio en Europa mediante compañías distribuidoras como Mediateket con base en Noruega, donde el producto comienza a tener presencia en Suecia, Noruega y el Reino Unido

La misión de la empresa es proveer soluciones integrales para la automatización de canales de T.V. con arquitectura abierta y tecnología de punta a un costo accesible. En otras palabras, Fission visto como un producto, es una solución basada en software diseñado sobre hardware de última generación. Software que es capaz de proveer una operación totalmente automatizada, por medio de un flujo de trabajo adaptable a diversos sistemas ya existentes dentro de una estación de transmisión de video por cable o televisión abierta.

Esquema General de una Estación de Televisión

Para poder obtener una mejor comprensión de la solución integral que ofrece Fission Software, es necesario describir de manera general la forma en la que se manejan la mayoría de las estaciones de televisión. Dentro de una estación que maneja canales a gran escala existen cuatro departamentos relevantes:

- Departamento de Producción de Contenido
- Departamento de Ventas
- Departamento de Programación y Continuidad
- Control Maestro

El departamento de Producción de Contenido es el encargado de generar el material a ser transmitido a la audiencia. Dependiendo del canal y su rubro, el material puede consistir de programas educativos, programas de entretenimiento, programas en vivo, etc. En esta misma área se encargan de elaborar la promoción interna de los programas que transmite la televisora.

El departamento de Ventas es el encargado de vender los espacios comerciales que existen dentro y entre los programas a transmitir, ya sea por medio de cortes comerciales, menciones dentro de los programas, video superpuesto, entre otras formas de comercialización. Una vez que el área de Producción de Contenido y el de Ventas han determinado el contenido y comerciales a transmitir, el departamento de Programación y Continuidad arma la pauta a ser trasmitida, la cual posteriormente es transferida al área de Control Maestro.

En Control Maestro se procesa la pauta que indica lo que se debe transmitir. Por ello, en esta área es donde se recaba todo el material audio-visual de diversas fuentes como: Producción de Contenido, estudios, clientes externos para obtener su comercialización, señales satelitales, etc., para finalmente poder reproducir el video y audio previamente planeados y así ser transmitido al teleauditorio.

En el área de Control Maestro es donde reside principalmente la solución que ofrece Fission Software a sus clientes.

Módulos de Fission

Fission siendo una solución modular, está formado principalmente por 4 módulos de software: Feed Capture Module, Capture Module, Management Module e Insertion Module. En este escrito para referirse a cada módulo, desarrollado por la compañía, se utilizará solamente su nombre, utilizando la palabra módulo de forma implícita, para facilitar la lectura. Dichos módulos se describen brevemente a continuación:

Feed Capture Module: Permite al usuario programar y realizar capturas de material audiovisual en un horario, periodo y duración predefinida; también cuenta con las herramientas para segmentar dicho material en bloques que faciliten al usuario su manejo desde una fuente satelital. Por medio de este módulo se puede retrasar la transmisión de la señal de entrada por un lapso predefinido para obtener repeticiones desfasadas de las señales. También provee herramientas de edición a corte para material audiovisual tanto a nivel archivo como lógico.

Capture Module: Facilita al usuario la ingesta de material audiovisual al sistema por medio de listas de captura que automatizan el vaciado de las cintas desde una videocasetera (VTR). Mediante las herramientas de edición a corte que ofrece, permite ligar el material audiovisual recién capturado con las entidades lógicas creadas desde otros módulos para ser transmitido posteriormente por el módulo Insertion. También provee herramientas para segmentar de manera lógica los programas, de acuerdo a los bloques comerciales asignados, para su reproducción al final de la cadena del sistema sin tener que modificar el archivo original de captura.

Management Module: Permite al usuario la creación, modificación y eliminación de elementos lógicos, que son ligados a la media audiovisual y utilizados para realizar pautas de transmisión y reproducción dentro del sistema. También permite crear reportes establecidos por el usuario de dichas actividades y la comunicación con otros sistemas con el fin de intercambiar pautas y reportes de transmisión. Asimismo, permite administrar y dar mantenimiento a los recursos de almacenamiento con los que cuenta el sistema instalado, realizando tareas de prevención de errores y depuración.

Insertion Module: Su función principal es ejecutar listas de reproducción de archivos de audio y/o video previamente establecidas en el sistema por medio del módulo Management o un sistema de tráfico externo, así como realizar edición y adición de material sobre éstas. También permite agregar efectos de superposición, edición y mezcla sobre los elementos de su lista. Además, provee la funcionalidad para insertar y ejecutar comandos pautados sobre dispositivos remotos a través de protocolos de comunicación como TCP/IP o por puertos del sistema como el serial o paralelo, de acuerdo a las necesidades del cliente. El módulo es capaz de reproducir el material de hasta cuatro canales simultáneamente en un mismo servidor.

Flujo de Trabajo con Fission

Una estación de televisión que utiliza Fission como su sistema de automatización dentro del departamento de Control Maestro para la transmisión de su señal de audio y video, trabaja descrito en una forma muy general, de la siguiente manera:

El flujo comienza en el módulo Management, dando de alta la lista de reproducción para cierto día. Ésta puede ser creada importando una pauta generada desde un sistema de tráfico externo o con la herramienta de creación de pautas incluida en el módulo.

Después, se ligan los archivos audiovisuales a transmitir con sus contrapartes lógicas creadas, en la base de datos, desde el Management, ya sea por un proceso de creación manual o por la importación de la lista de reproducción. Los elementos a ligar pueden ser: programas, versiones (cortes comerciales y promocionales), y eventos secundarios (gráficos, video y audio de superposición).

Los programas y versiones, por lo general, son capturados por medio del Feed Capture o Capture, aunque en ciertas estaciones la media ya ha sido digitalizada desde sus departamentos de producción, en este caso, basta con ubicar los archivos dentro del almacenamiento de Fission y ligarlos a sus respectivos elementos lógicos.

El diseño del sistema posee una gran flexibilidad en la administración del almacenamiento, ya que permite almacenar el video digitalizado en los discos locales de los ordenadores donde se captura el mismo, en el almacenamiento de los ordenadores que reproducen dicho material al aire o en sistemas de gran capacidad y disponibilidad como lo son una NAS o una SAN para guardar la multimedia empleada en el sistema.

Una vez que la lista de reproducción se encuentra lista y completa, ésta es reproducida en el Insertion que permite realizar en tiempo real modificaciones a la programación mediante la inserción de contenido y eventos secundarios de último momento.

Finalmente, una vez transmitido el material, el Insertion crea reportes de transmisión dentro de la base de datos de Fission. Estos reportes se cotejan a través del Management contra las pautas o listas de reproducción originales de forma automatizada para ubicar las diferencias que existen entre lo esperado a transmitir contra lo que se transmitió realmente y mantener un control de calidad en la señales de televisión.

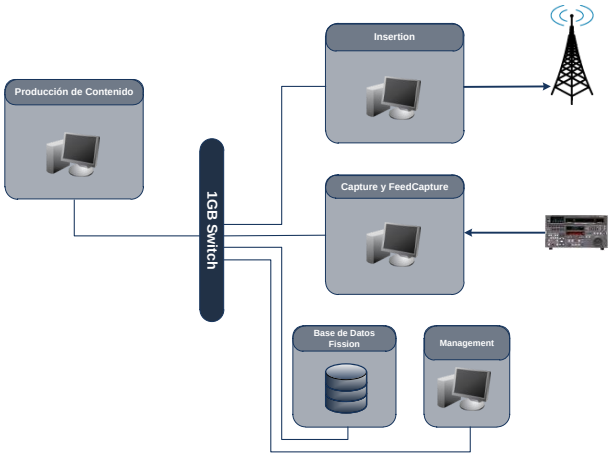


Figura 1.1- Esquema general sistema Fission

Capítulo 2 - Actividades dentro de Fission Software

En un inicio cuando ingresé a la compañía en el año 2006, ésta había logrado una venta importante con la distribuidora de contenido VISAT, actualmente conocida como Televisa Networks. En ese momento mi trabajo consistía en instalar la red física, los servidores dentro de su sitio y el software correspondiente a cada servidor de acuerdo al producto que había comprado el cliente. En la mayoría de los casos se trata de una red con topología de estrella, ya que el almacenamiento se trata de una SAN (Storage-Area Network) la cual es un sistema de almacenamiento masivo centralizado y requiere que un servidor actúe como MDC (Meta-Data Controller) el cual administra el bloqueo de archivos, la asignación de espacio y la autorización de acceso a datos de la SAN. Por lo que el trabajo en campo consistía en cablear la red tanto de fibra óptica como de cobre en las instalaciones del cliente. Colocar los servidores en su estante, instalarles el sistema operativo y el software que les corresponde de acuerdo al diseño de la instalación solicitada por el cliente.

En ese momento mi trabajo consistía de diversas tareas como: Instalar la red física, colocar los servidores en su sitio, instalarles el software correspondiente y configurarlo de acuerdo a la solución que había solicitado el cliente. En la mayoría de los casos la red a instalar cuenta con una topología de estrella, debido a que el almacenamiento se trata de una SAN (Storage-Area Network) la cual es un sistema de almacenamiento masivo centralizado y requiere que un servidor actúe como MDC (Meta-Data Controller) el cual está encargado de administrar el bloqueo de archivos, asignar el espacio de escritura y autorizar el acceso a datos de la SAN por clientes externos. Por lo tanto primero cableaba la red tanto de fibra óptica como de cobre en las instalaciones del cliente. Esta tarea consistía en extender los cables por el inmueble del cliente y crear las puntas de conexión de los cables, así como configurar los *switches* de red necesarios. Una vez establecida la red se instala el sistema operativo y el software a los ordenadores de acuerdo al diseño de la instalación solicitada por el cliente.

Una vez que se cuenta con la red instalada y configurada apropiadamente, el trabajo consiste en observar y analizar la estabilidad del sistema dentro del sitio. Como parámetros de estabilidad consideraba si existía algún error en el software, registraba en que ambiente y situación se ocasionaban por medio de herramientas desarrolladas internamente en la compañía. También otro dato a observar era el tráfico de red por medio de herramientas como WireShark. Una vez que el equipo se encontraba totalmente instalado y listo para ser utilizado, mi trabajo consistía en capacitar al personal de la estación que iba a estar a cargo del manejar del software dentro del área de Control Maestro. Los cursos impartidos consistían en explicar la manera más adecuada en que debían configurar y utilizar el software. Una vez terminados los cursos, mi trabajo se basaba en permanecer dentro de la estación como personal de apoyo para aclarar cualquier duda sobre el manejo del software a los operadores.

Este tipo de tareas son necesarias dentro de una instalación debido a que el producto se trata de un sistema de misión crítica que requiere de su correcto funcionamiento en todo momento para que la transmisión de la señal de televisión se realice.

Ya que me encontraba familiarizado con la forma de trabajo y las necesidades que se tienen dentro de una estación de televisión, la compañía me asignó trabajos de investigación para sondear las necesidades de los clientes dentro de las estaciones, así como averiguar si el cliente se encontraba satisfecho con el software e incrementar la calidad del servicio que ofrece la compañía. Se me asignó este tipo de trabajo debido a mis conocimientos dentro del área de desarrollo de aplicaciones y consideró que esos conocimientos me permitirían escribir reportes más comprensibles al equipo de desarrollo, ya que podría utilizar lenguaje técnico más específico con un enfoque a la programación del software y así poder cumplir con mayor rapidez las necesidades y exigencias del cliente.

Más adelante, y hasta la fecha, se me solicitó programar en sitio controladores (*drivers*) de algunos dispositivos que requieren compatibilidad con Fission. Debido a que Fission es un sistema abierto, en casi todas las ocasiones, los clientes desean que el sistema sea compatible con equipo con el que ya cuentan o compraron independientemente al producto que compraron por parte de Fission. En la mayoría de los casos, éste equipo de trata de *routers* o *switches* de señales de audio y video. Dado que la empresa cuenta con un sistema modularizado para controlar dispositivos externos, mi trabajo se remitía a implementar los protocolos propietarios de la compañía fabricante de cada dispositivo sobre el medio que éste requería, por ejemplo, ya sea sobre una conexión RS-232 ó TCP/IP. Por mencionar algunos ejemplos de los protocolos que he implementado hasta la fecha se encuentran el Intelligent Interface para el secuenciador Channel Box de Chyron, el protocolo Quartz para el ruteador de señal EQX de Evertz, la implementación de CSL de VizRT para controlar el secuenciador MultiChannel y el protocolo ESAMII para diversos secuenciadores y switches fabricados por Sony

También en diversas ocasiones tenía como trabajo realizar búsquedas de software especializado que agilizara o ayudara al cliente para aceptar la adaptación del uso de Fission Software dentro la estación. Para ejemplificar un caso, en una ocasión se me solicitó un software que agilice el copiado de archivos entre diversos servidor con una SAN o NAS. Teniendo en cuenta que esta necesidad era general entre los clientes, Fission decidió implementar una solución que se integrara con el sistema, a la cual llamó Fission File Transfer. Proyecto en el cual también colaboré en el desarrollo para administrar el copiado de archivos dentro de una red de Windows y así limitar la carga de la tasa de bits en la transferencia evitando que exista una interferencia en el streaming de los archivos de audio y video para su transmisión al aire.

En otro caso, un cliente dentro de una sala de bloqueos, donde la operación requiere el remplazar los cortes comerciales por comercialización local ya que la estación es una repetidora de señal. Requerían de un cronómetro que los usuarios pudieran lanzar y estar atentos a él en todo momento para saber cuanto tiempo les quedaba disponible, por lo tanto se les recomendó utilizar XNoteStopwatch ya que este permitía el ser iniciado y finalizado por medio de un pulso en el puerto serial RS-232, permitiendo automatizar toda la operación desde un servidor central en la sala de bloqueos.

Cuando la compañía determinó que me encontraba los suficientemente familiarizado con el ambiente de la televisión, me encomendó la tarea de desarrollar el sistema de monitoreo de señales de audio y video para la corroboración de la calidad y un

testigo de la señal transmitida. Debido a que en los últimos años me dediqué en gran parte de mi tiempo a la elaboración de este software, consideré pertinente ubicar una descripción del desarrollo del mismo como el objetivo principal de este escrito el cuál explicaré a detalle en el capítulo 3.

Durante el fin de una de las iteraciones del desarrollo del Fission Monitor se me invitó a colaborar con el desarrollo del sistema denominado Fission Archive. Éste es un sistema de archivo que tiene acceso a diferentes áreas dentro de la estación de televisión y provee las herramientas necesarias para crear catálogos con términos consistentes entre los usuarios. Los catálogos están compuestos por estructuras de metadatos, valores de los datos, reglas de validación y los archivos. De esta forma se le facilita a un usuario buscar la información y/o archivos que requieren. Principalmente está basado en un sistema de datos ubicados en una base de datos relacionados con archivos dentro de un sistema de almacenamiento masivo. En una ocasión un cliente, de acuerdo a sus necesidades, requería de un tipo de dato el cuál no estaba contenido en el sistema. El tipo de dato se trataba de un *árbol de archivos* dicha estructura consistía en carpetas las cuales a su vez podían contener carpetas y archivos de cualquier tipo. Mi labor fue implementar la creación de la estructura de la base de datos, la importación dentro del sistema de almacenamiento masivo y la extracción desde el mismo.

Recientemente, al momento de realizar este escrito también se me ha invitado a colaborar con el desarrollo de controles personalizables de Windows para poder entregar una GUI de última generación que facilité la tarea de inserción de comerciales y la administración y captura de multimedia en el área de Control Maestro. Mi última tarea consiste en investigar y desarrollar nuevos controles que faciliten el acceso a la información contenida en el sistema. Para ello me encuentro actualmente investigando acerca de menús contextuales como lo son los circulares.

En resumen, este es un listado de las tareas principales en las que he participado dentro de mi trabajo en la compañía Fission Software de México S.A. de C.V. desde mi ingreso hasta la fecha.

Capítulo 3 - Fission On-Air Monitor

Es importante notar que al final del flujo de trabajo que ofrece la compañía como solución, existe una necesidad a satisfacer dentro de las estaciones de televisión: la posibilidad de poder cotejar de manera auditiva y visual las diferencias que lleguen a existir en la transmisión con respecto a la lista de reproducción o contra la pauta inicial.

La compañía contempló dicha oportunidad de mercado como un módulo a agregar, dentro del sistema del área de Control Maestro, desde el cuál se podría ofrecer videoclips que contengan lo ocurrido durante la transmisión al aire a los otros departamentos para su revisión y análisis.

Afinando el concepto, el módulo que consideró a desarrollar tenía que ubicarse al final de la cadena como se muestra en la figura siguiente:

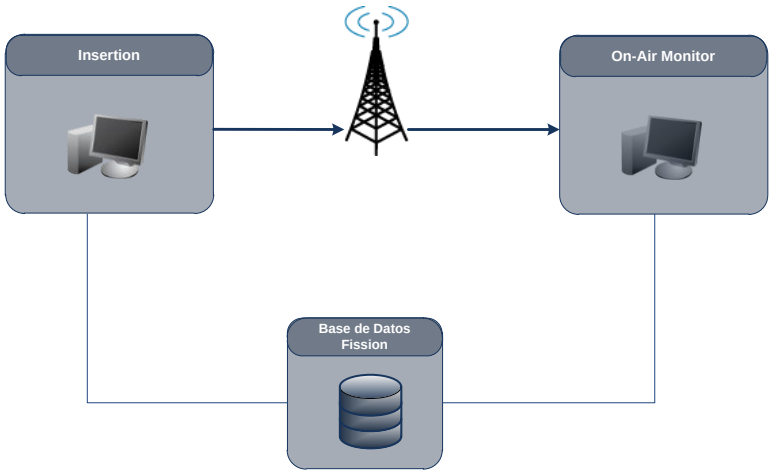


Figura 3.1 - Integración Fission OAM

Es importante resaltar que el módulo debe de estar ubicado en el flujo, posterior a la señal de retorno de la televisora. De esta forma el producto no depende del resto de los módulos de Fission, con lo que es posible instalarlo dentro de una televisora a pesar de que no cuente con Fission como sistema de automatización dentro del Control Maestro.

Basándose en esa necesidad, la empresa me encomendó desarrollar un sistema que grabe continuamente de manera digital las señales de audio y video transmitidas por una estación de televisión. Las grabaciones deben realizarse sin interrupción. La calidad de la señal capturada debe ser menor a la transmitida por la estación de televisión para cumplir la función de testigo y poder almacenar grandes tiempos de grabación aprovechando la compresión de video empleada.

El sistema debe de cumplir con los siguientes requerimientos:

- Grabación ininterrumpida de la señal de retorno de la estación de televisión
- Realizar un mantenimiento autónomo a las unidades de almacenamiento que utilice para los archivos de grabación
- Ser capaz de recuperarse automáticamente en caso de error
- Realizar búsquedas basadas en tiempo, fecha y canal dentro de sus registros de captura
- Realizar exportaciones a un formato de video digital que sea visible en la mayoría de los ordenadores sin la necesidad de instalar decodificadores adicionales.

A su vez, el sistema debe ser compatible con el sistema operativo Windows XP y posterior. El hardware especializado a utilizar para realizar la tarea de captura de las señales de audio y video es la tarjeta Optibase MM400 la cual se programa por medio de su paquete de desarrollo de software (SDK).

3.1 Requerimientos del Producto

Los requerimientos del sistema *Fission On-Air Monitor* se levantaron en base a las necesidades de un cliente con una importancia relevante para la compañía. El cliente requería un sistema autónomo autosustentable que cumpliera con la función de *testigo* de sus señales de audio y video transmitidas, para ser revisadas en caso de error o ser cotejadas en caso de duda con respecto al resultado del flujo de trabajo del área de Control Maestro.

El departamento de ventas de Fission Software, junto con su departamento de operaciones y desarrollo, realizó un primer acercamiento con el cliente basado en un bosquejo de funcionalidad básica de la interfaz del software basada en diálogos y una descripción de los eventos que ocurrirían al interactuar con ella. La propuesta, presentada al cliente incluía la decisión de desarrollar el sistema basado en el hardware que ofrece Optibase Inc., la tarjeta grabadora *Optibase MM400*, porque ésta cumple con: los estándares de calidad requeridos, un SDK sencillo de utilizar que ayuda a la pronta elaboración del sistema y un costo atractivo para el cliente.

3.1.1 Primer Bosquejo de la Interfaz

Fondo de pantalla

Al iniciar el programa se debe mostrar el diálogo que se muestra en la Figura 3.2. El diálogo debe de abarcar en su totalidad la resolución de un monitor de 1280x1024 pixeles. El nombre asignado a este diálogo es *fondo de pantalla* de la aplicación.

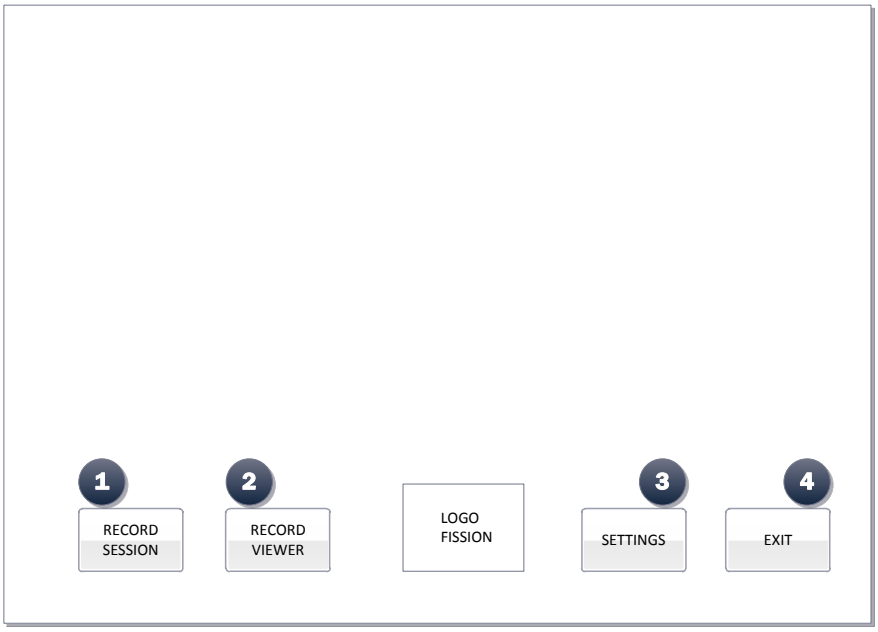


Figura 3.2 - Fondo de pantalla

El diálogo incluye cuatro botones que cumplen con las siguientes tareas:

1. Abrir una sesión de captura de una de las entradas del hardware de captura
2. Abrir una sesión de visualización del registro de captura

3. Abrir una ventana de configuración del sistema
4. Cerrar la aplicación.

Descripción de los eventos

Esta ventana ocupa toda la pantalla para evitar que el usuario pueda realizar otra tarea en el servidor mientras esté utilizando el software. Desde esta interfaz se puede iniciar una sesión de captura si se presiona el botón "*Record Session" (Sesión de Captura)*, posteriormente determina la entrada sobre la cual desea realizar la captura, mediante un diálogo intermedio de selección de entrada como el que se muestra en la Figura 3.3.

Como máximo se pueden abrir seis sesiones simultáneas de captura debido a que en un servidor se pueden instalar máximo tres tarjetas grabadoras que cuentan con dos entradas cada una.

Figura 3.3 - Diálogo selección de canal

Desde el diálogo principal también es posible iniciar una sesión de visualización del registro de captura presionando el botón *"Record Viewer" (Visualizador de grabación)*. De la misma manera que la sesión de captura, la visualización comienza con la selección del canal por medio de un diálogo que muestra el listado de los canales definidos en el sistema.

A cada entrada del sistema le es asignado un nombre de canal el cuál será utilizado como identificador único para una sesión de grabación, por lo tanto es importante considerar la restricción de **no repetir el nombre del canal a través de las diferentes entradas dentro del mismo servidor**.

El número máximo de visualizadores está delimitado por el hardware que albergue al sistema. Debido a que los visualizadores están implementados en su totalidad por medio de software, es necesario que se realicen pruebas posteriores al desarrollo para cuantificar la cantidad de recursos que ocupa cada instancia del visualizador y así poder determinar la cantidad límite de visualizadores simultáneos.

Finalmente, dando clic sobre el botón *"Settings"* (Ajustes) debe abrirse un diálogo que permita configurar la captura de cada entrada, la manera en la que se despliega el video en la entrada física, definir cuáles son las unidades de almacenamiento a utilizar, así como la manera en que se les dará mantenimiento.

Todos los diálogos descritos deben de aparecer dentro del fondo de pantalla y en ningún momento deberán cubrir el área de botones ubicados en la parte inferior.

Sesión de captura

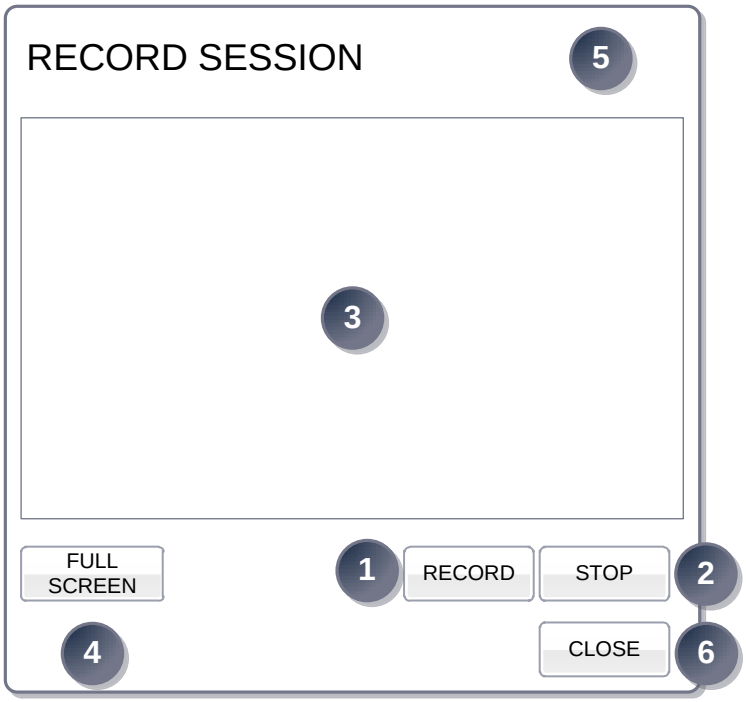


Figura 3.4 - Diálogo de sesión de captura

Los controles del diálogo de sesión de captura deben de cumplir con los eventos descritos a continuación:

1. Iniciar la captura de las señales de audiovisuales
2. Detener la sesión de captura
3. Mostrar del video de la señal de entrada en un recuadro
4. Mostrar el de la entrada en toda la pantalla
5. Reproducir el audio de la captura al seleccionar la sesión
6. Cerrar la sesión.

Descripción de los eventos

Al presionar el botón *"Start" (Comenzar)* debe comenzar a grabar en archivo las señales audiovisuales de la entrada correspondiente a la sesión de captura seleccionada.

Al presionar el botón *"Stop" (Detener)* se detendrá la sesión de captura en curso.

En el recuadro ubicado en la parte superior se debe **mostrar el video** con el que está siendo alimentada la entrada correspondiente **a una resolución de 320 x 240 pixeles** en caso de tratarse de una relación **4:3 o en una resolución de 320x180** en el caso de que se desee desplegar el video escalado **en una relación 16:9** en caso de que la fuente sea anamórfica.

Cuando este diálogo sea seleccionado por medio de un clic del ratón el audio de entrada correspondiente debe ser el único en salir por el dispositivo de salida de audio por defecto del sistema.

En este mismo diálogo se debe de mostrar de manera automática el porcentaje de espacio libre en la unidad de almacenamiento donde se capturan los archivos multimedia. La cifra debe mostrarse en porcentaje y actualizada cada 10 segundos.

Finalmente, al presionar el botón *"Close" (Cerrar)* se cierra la sesión de captura. Si la captura se encuentra activa, debe de aparecer un diálogo de aviso al usuario indicando que la sesión no puede ser finalizada al menos de que sea detenida la captura.

Sesión de visualización

En un diálogo donde se visualiza el material capturado se deben considerar los siguientes eventos con relación a los controles indicados en la Figura 3.5.

1. Buscar dentro del registro de archivos el audio y video correspondientes a la fecha y hora seleccionados
2. Reproducir en un recuadro el video correspondiente a la búsqueda realizada y el audio a través del dispositivo de salida seleccionado por defecto en el sistema operativo
3. Llevar a cabo las acciones básicas de un control de reproducción como se muestran a través de todo el sistema (Reproducir (Play), pausar (Pause), retroceder un cuadro (Step Back), avanzar un cuadro(Step Forward), rebobinar (Rewind), reproducir rápidamente(Fast Forward)).
4. Exportar el videoclip basado en los tiempos de inicio y fin dentro de los tiempos existentes del registro.

Figura 3.5 - Diálogo de visualizador de video

Descripción de los eventos

Al presionar el botón *"SEEK" (Búsqueda)* se realiza una búsqueda dentro del registro basado en la fecha y hora ingresados en los controles correspondientes.

La búsqueda puede tener un margen de error de hasta 3 segundos al mostrar el resultado, este valor puede ser configurado por el usuario. Cuando la búsqueda tenga éxito, se debe mostrar al usuario el video pausado en el cuadro correspondiente a la fecha y tiempo solicitados, junto con el nombre del archivo que contiene el video.

Cuando el video se esté reproduciendo, se debe desplegar en todo momento la fecha y hora correspondientes al video mostrado en el recuadro de la visualización.

Otro punto a considerar es que al momento de dar doble clic sobre el recuadro donde se muestra el video, éste se debe mostrar en toda la pantalla, abarcando toda el área del monitor sin perder la proporción de las dimensiones del video. En caso de que el video no recubra en su totalidad el área del monitor debido a su relación, se pintará en negro el área desocupada por el video.

Los controles de reproducción deben cumplir con su función al dar un solo clic sobre ellos. Los botones de [*Fast Forward*] y [*Rewind*] tendrán una funcionalidad extra. Ya que al ser presionados más de una vez, se irá incrementando la velocidad de la reproducción hacia adelante o hacia atrás dependiendo del caso. ***Las velocidades a soportar son 2X, 4X, 8X, 32X y 64X****, donde X es el número de veces contra la velocidad normal (1X) de reproducción.*

Configuración del sistema

El diálogo de configuración del sistema debe estar basado en pestañas, donde cada pestaña corresponde a cada una de las entradas presentes en el servidor.

Los controles del diálogo mostrado en cada pestaña serán asignados para cada uno de los ajustes a configurar en como se muestra en la figura:

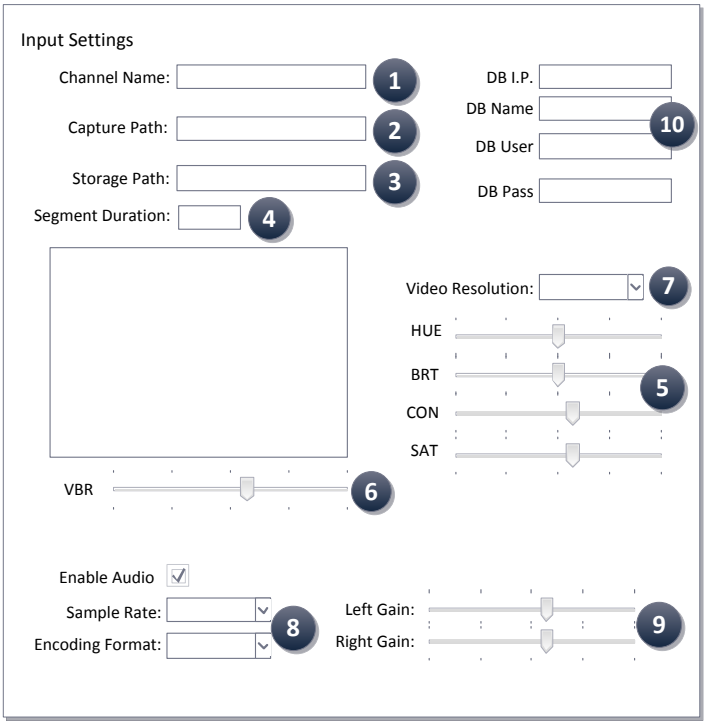


Figura 3.6 - Diálogo de configuración de entrada

1. Nombre del canal asignado a la entrada
2. Directorio en el cual se realiza la captura (Debe ser un directorio local)
3. Directorio del almacenamiento donde se guarda la captura (Puede ser un directorio local o remoto)
4. Duración del segmento
5. Valores de color (HUE), saturación (SAT), brillo (BRT) y contraste (CON)
6. Tasa de bits (*bitrate*) de grabación
7. Resolución de la grabación del video
8. Muestreo y formato de compresión del audio
9. Ganancia de audio por canal de audio
10. Conexión con la base de datos del sistema Fission.

Descripción de los eventos

Los cambios realizados dentro de los diálogos toman efecto al momento de dar clic sobre el botón "*Ok".* Una vez que se hayan actualizado las configuraciones de cada entrada el diálogo se cierra. Algunos de los cambios pueden tomar efecto durante la manipulación de los controles como son todos los valores contenidos en el punto 5, para su visualización instantánea.

En caso de presionar el botón *"Cancel" (Cancelar)* se cierra el diálogo sin guardar los cambios realizados durante la sesión de configuración.

De acuerdo al manejo de configuraciones del resto de los módulos de Fission, los ajustes de la configuración son guardados en el registro de Windows.

3.2 Diseño del Producto

De acuerdo a los requerimientos levantados por los ingenieros de campo, consideré que la mejor metodología para realizar el diseño era la llamada Arriba-Abajo (Top-Down), en donde ataqué el problema desde la interfaz gráfica hacia el núcleo del sistema. Por esa razón, una vez establecida formalmente la interfaz gráfica con el cliente a través de los requerimientos y mediante su abstracción, pude definir las tareas a ejecutar en la siguiente lista:

1. Capturar las señales de audio y video en un archivo de cada entrada física
2. Configurar el sistema
3. Mostrar al usuario las señales de entrada de la tarjeta
4. Buscar dentro de los registros de captura
5. Visualizar el contenido ubicado en los registros de captura
6. Exportar segmentos del registro de captura
7. Administrar los archivos ubicados dentro del registro de captura y el espacio libre de las unidades de almacenamiento.

Si se analiza con detalle las tareas mencionadas, se puede apreciar que la captura de señales A/V debe ser persistente en todo momento, es decir, no puede ser detenida para ejecutar cualquiera de las otras tareas, por lo que consideré que el paralelismo dentro del sistema es inevitable.

Otro punto importante que surgió mientras realizaba el análisis previo de las herramientas que debía utilizar, es la diferencia de codificaciones en las que se encuentran compilados el SDK de Optibase y las aplicaciones desarrolladas en Fission, donde el SDK de la tarjeta de captura está compilado en Multi-byte y en Fission se compilan en Unicode.

Los términos, Multi-byte y Unicode, se refieren a la forma en que se codifican las cadenas de caractéres al momento de compilar un programa o una biblioteca. La codificación Unicode es utilizada para el soporte de más idiomas internacionales con caracteres especiales como: el Chino, el Japonés o la escritura Cirílica, mientras que la codificación Multi-Byte sólo se limita a la mayoría de las lenguas romances en términos generales.

El hecho de que la herramienta de desarrollo de Optibase y las aplicaciones dentro de Fission estén compiladas en diferentes codificaciones impide que dentro de un proyecto, se incluyan ambas bibliotecas, lo que me orilló a separar en diferentes programas la capa de interfaz gráfica de la capa de manejo de la tarjeta.

Consecuentemente, la arquitectura Cliente-Servidor me resultó atractiva para la implementación del sistema solicitado.

3.2.1 Arquitectura Cliente-Servidor

La arquitectura Cliente-Servidor abreviada como **C/S**, es un modelo de diseño para sistemas que reparten el procesamiento entre un programa que realiza peticiones, denominado Cliente, a otro programa que le da respuesta, llamado Servidor.

Las ventajas principales de la arquitectura C/S son de tipo organizativo. Una ventaja que obtuve al utilizar este esquema de diseño, es que pude separar las responsabilidades de cada proceso, clarificando el diseño del sistema. La separación

entre cliente y servidor es de tipo lógico, es decir, el servidor no se ejecuta exclusivamente en una máquina diferente a la que alberga el cliente y tampoco se refiere a que el servidor esté forzosamente compuesto por un solo programa. Al contrario, por lo general dentro de este modelo, el servidor se descompone en diversos programas que pueden ser ejecutados de acuerdo a las necesidades del usuario, aumentando así el grado de distribución del sistema y convirtiendo de esta forma la manutención en una tarea más sencilla ya que en caso de error podría remplazar únicamente un módulo sin la necesidad de detener por completo la parte que actúa como servidor.

El cliente es el programa que expide una solicitud dentro de la arquitectura C/S, es decir, tiene el papel activo dentro de la comunicación y se encarga de solicitar la ejecución de los servicios ofrecidos por el servidor.

Figura 3.7 - Cliente-Servidor

Una de las ventajas que promete este diseño es la modularidad. Si se cuenta con un sistema lo suficientemente modularizado se pueden realizar modificaciones sustanciosas, ya sea del lado del cliente o el servidor, sin la necesidad de tener que modificar forzosamente la otra parte, en el caso de contar con un protocolo de comunicación diseñado convenientemente. A su vez, si se cuenta con un protocolo lo suficientemente robusto, facilita que el sistema se comunique con sistemas externos en caso de ser necesario.

3.2.2 Descripción General del Sistema

Considerando la arquitectura elegida y la abstracción de las tareas a implementar, resulta claro que la grabación de los archivos es la más importante de todo el desarrollo y fue necesario ubicarla en un módulo que funcione como servidor y esté totalmente separado del resto de las tareas. Este módulo también es capaz de configurar el sistema ya que este se encuentra en contacto directo con el hardware por medio del SDK de Optibase.

Una vez establecido el servidor me resultó lógico que las tareas de búsqueda y visualización del registro, estuvieran agrupadas dentro del módulo cliente el cual también muestra la interfaz de usuario del sistema.

La exportación de segmentos de video, o videoclips, del registro de captura, a pesar de que lo ubiqué dentro del diseño en el mismo lado del cliente, decidí desarrollarlo en un módulo distinto al de la interfaz. Esta decisión fue tomada con la finalidad de poder entregar un servicio de exportación, que sea capaz de ser actualizado con mejoras en caso de que, en un futuro, algún cliente lo solicite.

La tarea relacionada al mantenimiento de archivos y almacenamiento masivo debe de ejecutarse como un sub-módulo del servidor, porque solamente de esta manera se puede estar completamente seguro de que el módulo de mantenimiento tiene acceso a las unidades de almacenamiento asignados a la captura debido a que ahí se encuentra el hardware de captura. Ya que en el supuesto de que lo hubiera colocado en el lado del

cliente, no tendría el acceso a las unidades de disco si el módulo cliente se ubicara en un ordenador remoto.

Una de las razones importantes que me impulsó a ubicar el proceso de mantenimiento en un programa o módulo distinto al que ofrece el servicio de captura fue, que cualquier problema crítico, como un error fatal dentro del proceso de mantenimiento, podría llegar a interrumpir el proceso de captura, No obstante, ambos módulos comparten su configuración ubicada en el registro del sistema operativo.

Considerando el modelo arquitectónico elegido y la asignación de tareas, los nombres de los módulos se asignaron de la siguiente forma:

- Fission Monitor Server (Captura y configuración del sistema por llamadas remotas)
- Fission Monitor Client (Búsqueda, visualización y exportación)
- Fission Monitor Upkeeper (Mantenimiento de las unidades de disco)
- Fission Monitor Exporter (Exportación de videoclips)

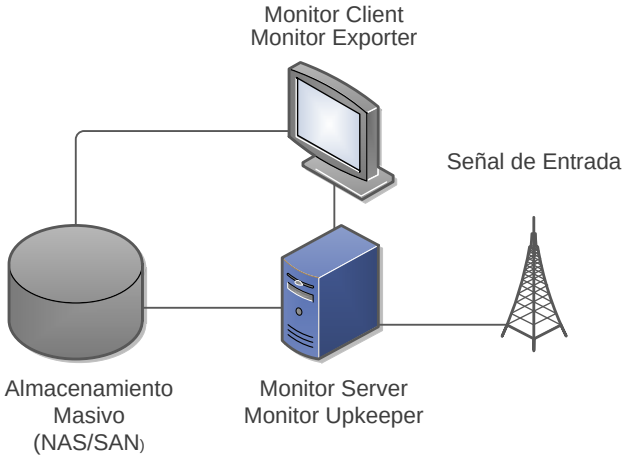


Figura 3.8 - Diagrama modular de Fission On-Air Monitor

Extendiendo la descripción de la funcionalidad de cada módulo, aclaro que el módulo Fission Monitor Server es el núcleo del sistema. Su propósito principal es el de capturar los archivos multimedia en un dispositivo de almacenamiento local (generalmente un disco duro) basado en reglas previamente establecidas por el usuario final.

Posteriormente, este módulo ejecuta periódicamente el módulo Fission Monitor Upkeeper, que está a cargo de la transferencia de archivos multimedia a una unidad de almacenamiento a largo plazo y a borrar los archivos dependiendo de su antigüedad en la unidad de almacenamiento y el espacio libre en la misma.

El módulo Fission On-Air Client es la interfaz gráfica del sistema que se presenta al usuario final. Desde sus controles se puede iniciar y detener una sesión de captura de las señales de audio y video. También permite una visualización previa de lo que se está capturando en tiempo real. Además, ofrece los controles que permiten configurar la captura de cada entrada independientemente de las otras. Igualmente, puede establecer las reglas de mantenimiento de las unidades de disco asignadas a la captura y al almacenamiento a largo plazo. En este módulo, el usuario puede buscar los archivos multimedia relacionados a una fecha, hora y canal en específico y exportar segmentos del registro de captura en el formato ASF. Para llegar a esta conclusión, me basé en una

investigación previa que me indicó que dicho formato puede ser reproducido en cualquier servidor que opere bajo Windows en cualquiera de sus versiones XP o posterior. Además, este formato permite codificar el video en una calidad lo suficientemente baja con lo cual disminuye su tamaño en bytes agilizando su transferencia por cualquier medio electrónico, dígase correo electrónico o transferencia FTP.

En la siguiente figura ilustro la interrelación de los módulos que implementé, donde solamente interactúan los hexágonos adyacentes:

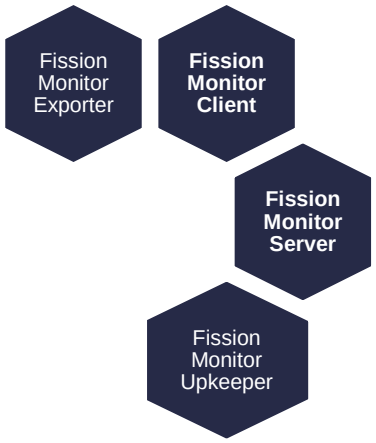


Figura 3.9 - Interacción de los módulos

3.2.3 Modelo Iterativo Incremental

El departamento de desarrollo de productos, dentro de Fission Software, implementó un esquema de trabajo definido empíricamente basado en el progreso y necesidades de la empresa durante su crecimiento. La forma en la que los fundadores de la compañía se las han ingeniado para cumplir con las soluciones requeridas, por la estaciones de televisión, tiene base en el *Modelo Iterativo Incremental* debido a la cantidad de personal y compromisos por entregar a tiempo con diversos clientes.

Es erróneo pensar en que “iterativo” e “incremental” son sinónimos dentro del desarrollo de software. Este error es común debido a que en la práctica ambos conceptos van de la mano. Cuando hablamos de un *Modelo Iterativo* nos referimos a que el desarrollo va en ciclos, mejorando la calidad del producto, sin embargo, a éste no se le añade funcionalidad. En el caso del *Modelo Incremental* en cada entrega al cliente, se van añadiendo fragmentos de funcionalidad al producto a partir de la primera entrega que se denomina la *versión esencial* o *versión base* del producto.

En términos generales el *Modelo Iterativo Incremental* consiste en desarrollar un sistema a través de ciclos repetitivos (iterativo) y con porciones de funcionalidad en cada entrega (incremental). De esta forma los desarrolladores de software aprenden las necesidades específicas de los usuarios finales y los usuarios pueden tener una idea más clara y concreta de sus necesidades, a través del uso del sistema.

Es decir, en cada iteración existen cambios al diseño, mayor funcionalidad añadida al producto y correcciones a posibles errores que pudieran existir en la entrega anterior, de tal forma que las versiones van evolucionando hasta que el sistema quede totalmente implementado.

Figura 3.10 - Ciclo iterativo del modelo Iterativo Incremental

Este procedimiento consiste básicamente de una fase inicial, una fase iterativa y una lista de control de proyecto. En la fase inicial se lleva a cabo un análisis de todo el problema, posteriormente un diseño general de todo el sistema, junto con una calendarización del desarrollo del producto y terminando en la entrega de una versión base del mismo.

Durante la entrega de la versión base, el usuario está a tiempo de reaccionar y verificar que sus requerimientos fueron comprendidos totalmente. En caso de que ocurriera lo contrario durante esta etapa, no existen grandes repercusiones ya que se puede volver a la etapa de diseño. El rediseño de la aplicación se realiza con la finalidad de entregar una nueva versión base que cumpla las expectativas del usuario y a partir de la cual se irán entregando versiones mejoradas. Una vez que el usuario comienza a utilizar la versión base, el proyecto avanza a la fase iterativa. En ese momento entra en juego la lista de control de proyecto. En ella se lleva un registro de todas las tareas que deben ser realizadas para guiar la fase iterativa, de lo contrario esta forma de trabajo se puede volver muy confusa tanto para los desarrolladores como para el cliente, ya que sin una guía el sistema puede llegar a un punto sin fin donde los tiempos de entrega se vuelven ambiguos y ninguna de las partes comprende la problemática para la que está siendo desarrollado el sistema.

Por tanto, la Lista de Control de Proyecto debe de incluir las nuevas funcionalidades a ser desarrolladas, así como los puntos de rediseño que tuvieron que ser tomados para satisfacer las necesidades del cliente. Es importante que sea revisada constantemente durante la etapa de análisis en cada iteración.

La fase iterativa incluye el rediseño e implementaciones de nuevas capacidades del sistema que fueron añadidas a la Lista de Control de Proyecto como resultado del análisis de la versión actual. En una ejecución ideal del modelo, el rediseño e implementación en cada iteración debe ser sencillo, directo y modular. El análisis de una iteración está basado en la retroalimentación por parte del cliente y el análisis del sistema en sitio. Dicho análisis debe estar basado en el uso, eficiencia, estructura, modularidad y sobre todo en el alcance de las metas previamente establecidas de la iteración anterior.

Debilidades del modelo

Uno de los puntos débiles de este modelo de desarrollo es que el cliente debe estar involucrado en todo momento durante el desarrollo del proyecto. El rápido crecimiento que ha tenido la compañía Fission Software, en gran parte, se debe a que la mayoría de las estaciones de televisión trabajan todos los días del año cubriendo las 24 horas del día y están dispuestas a ir de la mano con la compañía para poder concretar una solución de automatización que les permita mejorar los procesos de producción de sus señales de televisión. Desafortunadamente, el costo de ejecución de dicho modelo implica para Fission Software mayores gastos de manutención, así como transporte y viáticos en horarios extremos para su personal.

Otro punto débil que se aprecia dentro de este modelo, es que la compañía que ofrece la solución requiere de profesionales con habilidades arriba del promedio y que se encuentren comprometidos a cumplir con las necesidades del cliente. Para poder cumplir con esta labor, Fission Software cuenta con personal capacitado dentro de las áreas de desarrollo y operaciones, que cuentan con una comunicación directa para resolver, con la mayor velocidad posible, problemas en el software cuando estos suceden en sitio. Para llevar una guía adecuada de los avances y metas a alcanzar se establecen planos de ruta para cada módulo del sistema que brinda las soluciones esperadas a los clientes.

Iteraciones del proyecto

Basado en el Modelo Iterativo Incremental la primera tarea a realizar es un análisis de los alcances que debe de tener el proyecto y plasmar las fases del desarrollo en una Lista de Control de Proyectos. Esta herramienta, por muy simple que parezca, permite a todas las áreas dentro de la empresa el intercomunicarse para generar una idea más concreta sobre que punto se encuentra el desarrollo, tener un mejor control sobre los tiempos de entrega y fungir como guía para los programadores ayudándoles a obtener metas más claras sobre el alcance. A continuación, se muestra una lista ejemplo realizada durante la etapa inicial del proyecto en discusión:

Tabla 3.1 - Lista de control de proyecto

Fission On-Air Monitor	
Lista de Control de Proyectos	
Fase	I-1
Versión a Entregar	1.0.0.0
Descripción	Entregar al cliente una versión base que cumpla con la funcionalidad mínima de grabar y visualizar la señal que se está grabando
Tareas	General: 1. Implementar Sockets 2. Diseñar y crear skins de programa cliente Módulo Servidor: 1. Implementar wrapper SDK de Optibase 2. Implementar protocolo de comunicación 3. Implementar servidor que interprete correctamente protocolo con cliente Módulo Cliente: 1. Implementar gráfica visualizadora de stream SDP por medio de DirectShow 2. Implementar Recrod Session 3. Implementar Settings 4. Implementar comunicación con el servidor.
Personal	Salvador Medina
Fecha de Entrega	26/08/2006

Fase I

Como se puede observar en la tabla, la *versión base* del producto tiene que ser capaz de grabar bajo una configuración establecida por el usuario para aprovechar el espacio en disco duro local. Las capturas de las señales de entrada deben de realizarse independientemente de acuerdo a las necesidades de cada canal. En esta fase el software debe de permitir al usuario visualizar la multimedia contenida dentro de los registros de captura. Una vez que se obtiene la versión base que incluye la funcionalidad mínima del sistema al concretar la fase I, el proyecto entra en la fase II.

Fase II

Durante la segunda fase del proyecto, el sistema debe ser capaz de exportar segmentos del registro, o videoclips, definidos por el usuario mediante la selección de fecha y hora tanto del inicio como del final del clip. Los videoclips deben de ser mostrados en un visualizador proporcionado por el programa cliente en la mayoría de los equipos sin la necesidad de instalar codecs extra al ordenador.

Fase III

En una tercera etapa del desarrollo, el sistema debe ser capaz de guardar la multimedia capturada en una unidad de almacenamiento masiva, como una NAS (Network-Attached Storage) o una SAN (Storage Area Network) y esta funcionalidad debe ser opcional para mantener la flexibilidad del sistema. Así mismo, debe de liberar espacio en disco mediante la eliminación de archivos en las unidades de almacenamiento designadas para la captura y para el almacenamiento a largo plazo.

Los parámetros que tomé en cuenta para realizar esta tarea son el espacio libre en la unidad de almacenamiento y la vida útil de los archivos. En caso de que el usuario defina un espacio libre mínimo requerido, se deberán borrar los archivos más viejos que estén ubicados en la unidad de disco. Si varios canales comparten la misma unidad de disco, se borrará el archivo más viejo de todos los canales. En caso de realizar un mantenimiento de disco por medio de expiración de archivo, simplemente se tiene que verificar en cada uno de los archivos contenidos que su tiempo de vida no sea mayor a la establecida por el usuario, el tiempo de vida del archivo está basado en la fecha y hora actual contra la fecha y hora de inicio de grabación del archivo.

El proceso de mantenimiento puede interpretado por el siguiente pseudocódigo:

```
1.  Obtener listado de cada uno de los directorios de captura y almacenamiento
2.  Ordenar cada uno de los listados obtenidos
3.  Agrupar las listas de acuerdo al disco en que están ubicados
4.  Por cada grupo obtenido
    4.1. Ordenar cronológicamente y mezclar las listas dentro del grupo
5.  Por cada lista obtenida de cada grupo
    5.1. Mientras no exista el espacio libre mínimo en el disco establecido por el usuario
        5.1.1.  Borrar el primer archivo de la lista
        5.1.2.  Eliminar el primer elemento de la lista
    5.2. Si no existen más archivos en la lista y no hay espacio libre disponible
        5.2.1.  Notificar error en el espacio libre correspondiente al disco del grupo
    5.3. Mientras el primer archivo de la lista tenga una vida mayor a la permitida por el
         usuario
        5.3.1.  Borrar el primer archivo de la lista
        5.3.2.  Eliminar el primer elemento de la lista
6.  FIN
```

Finalmente, como un comentario extra del diseño actual, un cliente distinto al que originó el proyecto solicitó que el sistema permita la visualización y exportación de segmentos en un ambiente multiusuario, en otras palabras, requería de un módulo nuevo que tenga acceso al registro de captura para buscar y exportar segmentos desde un módulo independiente. Gracias a la modularización descrita al inicio del capítulo, este programa lo pude implementar mediante la extracción del funcionamiento de la *Sesión de Captura* y reutilizando el módulo de exportación dentro de una nueva aplicación llamada Fission Monitor Viewer.

3.3 Herramientas Utilizadas para su Programación

En la compañía los programas compatibles con el sistema operativo Windows son desarrollados con la herramienta Visual Studio 2005 mediante el lenguaje VC++ utilizando las bibliotecas MFC (Microsoft Foundation Classes) que proporciona Microsoft.

Para desarrollar el sistema también utilicé DirectShow, herramienta proporcionada por Microsoft para la manipulación del audio y video.

Por último, fue imperativo que empleara el SDK que proporciona Optibase con su hardware para poder administrar el uso de la tarjeta encargada de capturar las señales de audio y video de acuerdo a los requisitos de la aplicación.

3.3.1 MFC

La biblioteca MFC encapsula algunas funcionalidades de la API de Windows en clases para el lenguaje de programación Visual C++, que facilitan el manejo de objetos, ventanas y controles de Windows.

Con esta herramienta se pueden desarrollar programas de una manera más rápida, ya que al momento de crear aplicaciones el código generado es mucho más fácil de escribir y de leer por todos los macros que éste incluye, teniendo como resultado proyectos de fácil mantenimiento.

VC++ posee un amplio uso de macros. Una macro es una definición o un texto breve que puede llegar a remplazar líneas enteras de código creando consecuentemente un lenguaje breve y conciso. Las macros que MFC provee se utilizan para el manejo de los mensajes o excepciones que arroja el sistema o aplicación. También son utilizadas para la seriación o creación de instancias de clases dinámicas. Microsoft creo las macros con la intención de reducir el uso de memoria evitando la creación de tablas virtuales como está estipulado dentro del estándar de C++. También lo creo con la intención de no tener que realizar un parseo una y otra vez.

En el caso particular de este trabajo utilicé la versión 8.0 de esta herramienta que está incluida en el programa de desarrollo Visual Studio 2005 SP1.

3.3.2 DirectShow

DirectShow es una API que proporciona Microsoft para la manipulación de archivos multimedia en diferentes versiones del sistema operativo Windows. Ésta fue añadida al SDK estándar de desarrollo de aplicaciones de Windows en el año 2007.

La arquitectura utilizada por DirectShow es del tipo modular, donde cada etapa del procesamiento es ejecutada por un objeto denominado filtro. DirectShow provee un conjunto de filtros estándar que pueden ser utilizados directamente por los programas, aunque esto no limita a los programadores a desarrollar sus propios filtros para aumentar la funcionalidad de DirectShow.

Para ilustrar el funcionamiento de DirectShow se muestra en la siguiente figura una gráfica sencilla encargada de reproducir un archivo de video con audio embebido:

En la figura que se muestra a continuación se observan componentes en forma rectangular, estos representan a los filtros. Cada filtro cuenta con entradas y salidas denominadas pines. Un pin de entrada solamente puede ser conectado a un pin de salida de otro filtro, generando así un flujo de datos dentro de la gráfica.

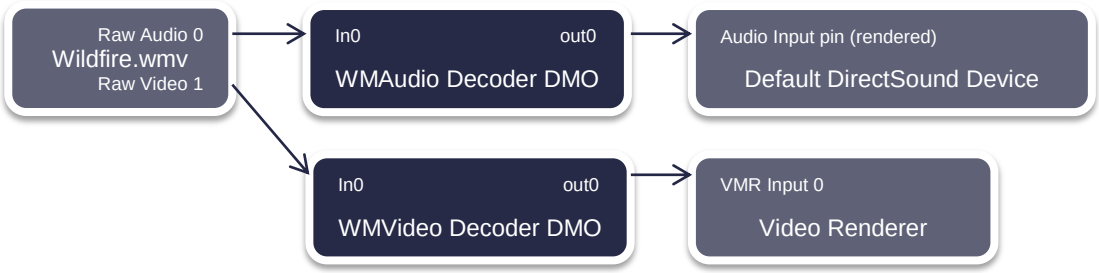


Figura 3.11 - Gráfica de ejemplo de reproducción de video

La arquitectura basada en filtros permite al programador reproducir, capturar o editar archivos multimedia de algunos formatos comunes como MPEG-1, MP3, WMA, WMV, MIDI en contenedores como ASF, AVI y WAV ya que los filtros requeridos están incluidos dentro del SDK.

Sin embargo, para el software en discusión era necesario poder manipular archivos codificados en MPEG-4, AAC contenidos en formato MP4, por lo que requería de filtros desarrollados por terceros, lo cual implicaba un costo adicional por licenciamiento por el uso de la patente y/o tecnología utilizada en el códec.

Esta herramienta me fue útil ya que me permitió desarrollar un reproductor y un editor de video de una manera muy sencilla.

Uso de FilterGraph

Al momento de programar una aplicación haciendo uso de DirectShow es importante tomar en cuenta el objeto *Filter Graph Manager,* que traducido al español se entiende como el *Administrador de Gráficas basadas en Filtros,* pero en futuras ocasiones haré referencia a este componente por su nombre en inglés.

Una de las funciones principales de este componente es establecer un reloj de referencia interno para coordinar los cambios de estado dentro de los filtros utilizados en la gráfica. Contando con una referencia interna es posible informar a la aplicación sobre los eventos ocurridos dentro la gráfica a la aplicación. También provee los métodos e interfaces necesarias para construir y manipular las gráficas. Toda aplicación que programe utilizando el *Filter Graph Manager* debo de seguir los siguientes pasos:

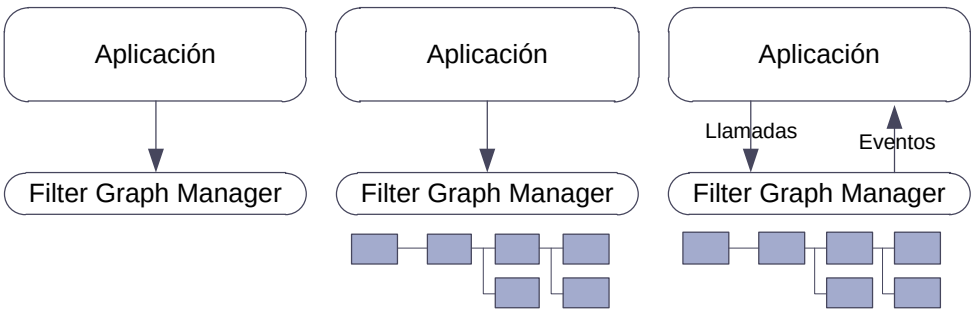


Figura 3.12 - Uso del Filter Graph Manager

1. Crear una instancia del *Filter Graph Manager*
2. Crear una gráfica por medio del *Filter Graph Manager*. Los filtros a utilizar los determina la aplicación de acuerdo a sus necesidades.
3. La aplicación utiliza al *Filter Graph Manager* como la interfaz para controlar la gráfica y flujo de datos que corre a través de los filtros. Durante este proceso la aplicación recibe eventos que indican el estado de la gráfica.

Durante el uso del *Fiter Graph Manager* fue importante que considerara la tecnología de Conexión Inteligente (Intelligent Connect). La *Conexión Inteligente* se refiere a los algoritmos que utiliza, internamente este componente, para construir todo o parte de la gráfica basándose en la compatibilidad de los filtros instalados en el sistema con los ya incluidos en la gráfica, combinando los filtros para intentar resolver la construcción de la gráfica.

Codec HELL y sistema de méritos en códecs

A pesar de que puede ser muy práctico permitir al administrador de gráficas elegir los filtros necesarios para completar una gráfica, esto puede llegar a ser un gran inconveniente para el usuario.

Cualquier fabricante de códecs dentro de la industria puede desarrollar sus propios filtros compatibles con DirectShow para poder codificar y decodificar diferentes tipos de multimedia.

Los filtros al momento de ser instalados son registrados con ciertos puntos de mérito en el registro del sistema operativo. Los puntos de mérito determinan la capacidad que tiene el filtro para poder trabajar sobre cierto tipo de multimedia en específico. En una instalación limpia dentro de un ordenador, este ambiente es el ideal para que los programas trabajen apropiadamente con códecs desarrollados por otras compañías.

No obstante, en la realidad, esta forma de trabajo trae consigo más errores que beneficios, ya que cualquier usuario puede instalar la cantidad de códecs que desee o considere necesario para su ambiente de trabajo y, tomando en cuenta, que todos los fabricantes al momento de crear el instalador de sus códecs tratarán de asignar el mayor número de puntos de mérito a sus filtros, ocasiona que se asignen los puntos de manera irregular dentro del ordenador, provocando que la gráfica creada por medio de la "Conexión Inteligente" del Filter Graph Manager tome filtros de diferentes fabricantes generando una gráfica disfuncional con filtros incompatibles, trastornando la finalidad de la Conexión Inteligente.

Para ejemplificar esta situación, se considera que un usuario instala 2 paquetes de códecs para decodificar y reproducir los archivos del tipo MPEG-4 versión 2. Considerando al primer fabricante como la compañía LEAD y al segundo siendo 3IVX. Los códecs de LEAD poseen 5 puntos de mérito para sus demultiplexores y decodificadores. En cuanto a los otros codecs, el paquete de instalación de 3IVX asigna 5 puntos de mérito a sus demultiplexores así como a sus decodificadores de video, sin embargo, asigna 3 puntos de mérito a sus decodificadores de audio.

Basándose en dicho supuesto, al momento de crear una gráfica para reproducir un archivo de video con audio embebido codificado en MPEG-4 versión 2 con audio codificado en AAC en formato MP4, la C*onexión Inteligente* del *Filter Graph Manager* daría como resultado una gráfica con filtros de diferentes fabricantes.

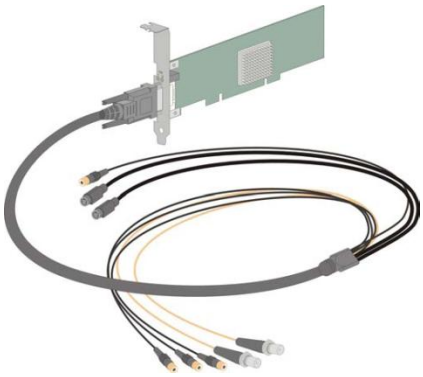
Este resultado en ocasiones podría ser algo positivo, si el decodificador de audio de LEAD funciona mejor que el de 3IVX. Sin embargo, en la práctica que he adquirido en las diversas instalaciones del producto, en la mayoría de las ocasiones esto resulta en

una gráfica disfuncional, donde el audio no esté sincronizado con el video o los decodificadores no sean del todo compatibles, construyendo una gráfica que reproduce sólo el video sin salida de audio.

Es por ello que se debe de tener mucho cuidado al momento de programar una gráfica mediante la *Conexión Inteligente* del *Filter Graph Manager.* Para validar si el ordenador cuenta con un ambiente apropiado para la reproducción de ciertos archivos multimedia, el uso de la herramienta *GraphEdit* me permitió visualizar la gráfica que se formaba al utilizar dicha tecnología y así podía verificar que el ambiente fuera propicio para que el módulo Fission Monitor Client funcionara correctamente.

3.3.3 Tarjeta Optibase MM400

La tarjeta MM400 tiene las dimensiones estándar de una tarjeta PCI de bajo perfil, en otras palabras, su altura y su longitud son de la mitad de una tarjeta estándar para una ranura PCI dentro de un ordenador resultando en una tarjeta con dimensiones de:



- 15.24 mm de ancho
- 175.26 mm de largo
- 106.68 mm de alto

Figura 3.13 - Tarjeta Optibase MM400 son su cable

La entrada de la señal de video es análoga al igual que la del audio. Cada tarjeta es capaz de capturar 2 señales de audio desbalanceado con su respectiva señal de video, ya sea compuesto o S-Video al mismo tiempo.

Para alimentar la entrada de la tarjeta con la señal, ésta cuenta con un cable conector que proporciona por cada entrada: un conector de S-Video, 1 conector BNC de video compuesto y 2 conectores de audio RCA por canal de entrada. El cable agrupa todas estas conexiones en un conector DVI que se conecta a la tarjeta.

El hardware requiere como mínimo para su instalación y uso un ordenador que cumpla con las siguientes características:

- Un procesador Pentium II a 450 MHz
- Una ranura PCI de 33 MHz, 66 MHz ó PCI-X (ya que la alimentación de la tarjeta es de 5V)
- Sistema Operativo Windows Server 2000 Service Pack 3 o Windows XP Service Pack 1
- 256 MB en memoria RAM
- Tarjeta gráfica con 128 MB de memoria en video
- Direct X 9.0 instalado en el sistema.

Las características técnicas de la tarjeta capturadora que utilicé pueden ser resumidas dentro de la siguiente tabla:

Tabla 3.1 - Características técnicas de la tarjeta MM400

Estándares Cumplidos	▪ ISO/IEC 14496-2:1999 (MPEG4 Video) ▪ ISO/IEC 14496-3:1999 and AMD1 2000 (AAC LC) ▪ MP4 Format - ISO/IEC 14496-1:2000(E), ISMA compliant
Señales Entrada	▪ Doble entrada
Entradas de Video (x2)	▪ Video Compuesto ▪ S-Video ▪ Formatos NTSC y PAL
Entradas de Audio (x2)	▪ Estéreo Analógico Desbalanceado (RCA) ▪ Impedancia mínima de 10 kΩ
Parámetros de Video	▪ Muestreo: 1- 30 FPS ▪ Bit rate: 8Kbps - 5Mbps ▪ Multi-stream: Capacidad de codificar cada entrada en 3 configuraciones distintas ▪ Detección de movimiento ▪ Detección de cambio de escena ▪ Segado horizontal de la imágen
Parámetros de Audio	▪ 24-320 Kbps MPEG-4 AAC LC ▪ Frecuencias de muestreo: 8, 11.025, 22.5, 32, 44 y 48 KHz
Formato de Archivos Capturados	▪ MP4 (audio) ▪ MP4 (video) ▪ MP4 (video con audio embebido)
Modos de Audio	▪ Mono ▪ Estéreo ▪ PCM lineal
Streaming de Red	▪ RTP sobre UDP ▪ Multicast/Unicast, cumpliendo el estándar ISMA
Hardware	▪ PCI de bajo perfil: 167.64 mm x 64.41mm ▪ Consumo de energía: ▪ 9 W a +5 VDC ▪ 5 W a +3.3 VDC ▪ 0.5 W a +12 VDC ▪ 3 tarjetas en paralelo

La siguiente tabla muestra las resoluciones que soporta la tarjeta:

Tabla 3.2 - Resoluciones de video soportadas por la tarjeta MM400

Resoluciones	NTSC	PAL
QSIF	160x112 176x112	176x144 192x144
SIF	320x240 352x240	352x288 384x288

También fue de gran importancia que tomara en cuenta las especificaciones con las que se generan los archivos de video y/o audio capturados por medio de la tarjeta. Ya que en toda ocasión los clientes requieren de un estándar en específico para este tipo de soluciones.

En el caso particular los archivos capturados son archivos en formato MPEG-4 ver. 2 cumpliendo con el estándar ISO/IEC 14496-2:1999 para la codificación de video y el audio embebido en el mismo archivo codificado en AAC.

3.3.4 Optibase SDK

Para que pudiera programar sobre la tarjeta MM400, Optibase proporciona la herramienta de desarrollo Optibase SDK. Dentro de la herramienta de desarrollo se incluye la clase `CRte` para el lenguaje de programación VC++, junto con proyectos que ejemplifican la forma de programar la tarjeta mediante el uso del paquete de desarrollo.

Los ejemplos muestran que al inicializar, de una manera específica y ordenada, una instancia de la clase CRte representa una de las entradas física de las tarjetas que se encuentren instaladas en el servidor. La inicialización es importante ya que ésta determina que entrada es la asignada. A su vez, ésta herramienta brinda suficientes bibliotecas que permiten el desarrollo sobre otro tipo de tarjetas producidas de la misma compañía.

Dado que mi objetivo era crear un programa que capture señales de A/V a través de la tarjeta MM400, pude sintetizar mediante un análisis del SDK, en términos generales, que los pasos a ejecutar para realizar una captura utilizando esta herramienta son:

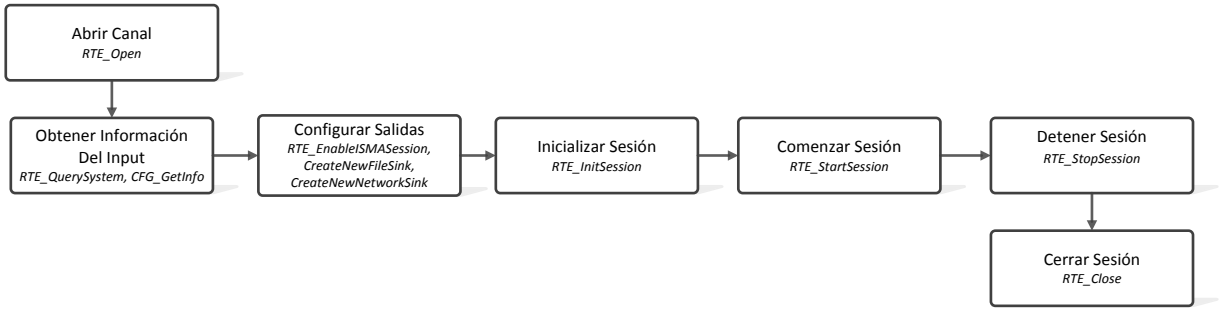


Figura 3.14 - Uso básico del Optibase SDK

Es bien sabido por los desarrolladores que en muchas ocasiones los SDK que proveen diversas compañías están acompañados de una mala documentación y un escaso soporte técnico para resolver dudas. Desafortunadamente este fue el caso, por lo que decidí documentar en esta sección una guía de los pasos básicos a seguir para concretar los diferentes procesos establecidos en el diagrama anterior.

Código de apertura de entrada

Para obtener la referencia de una entrada y comenzar a utilizarla: primero tuve que utilizar el método *RTE_Open*, obtener la referencia a *RTE_System* y de esta forma conseguir el *handle* de la entrada que decidí manipular:

```
bool OpenFirstInputAvailable(void)
{
   if( RTE_Open(NULL, &RTE_Handle) != 0 )
       return false;
   if( RTE_QuerySystem(RTE_Handle, &RTE_System)  != 0 )
   {
        RTE_Close(RTE_Handle);
        return false;
   }
   RTE_ConfigHandle = RTE_System->phCfg[0];
   return( RTE_ConfigHandle != NULL);
}
```

donde:
RTE_Handle es del tipo HRTE
RTE_System es del tipo RTESYS
RTE_ConfigHandle es del tipo HCFG

Secuencia de llamadas para inicializar una sesión de grabación y transmisión

Una vez que pude obtener la referencia a la entrada, el proceso que requería de un mayor cuidado dentro del flujo de trabajo era la inicialización de la misma, ya que ésta requiere del siguiente orden en específico:

1. Obtener la información de configuración por medio de **CFG_GetInfo**
2. Obtener estado de la entrada llamando **a CFG_GetCaps** y obteniendo un apuntador a la estructura *CFGCAPS*
3. Llamar **RTE_OpenConfig** para comenzar la secuencia de inicialización
4. Obtener todos los formatos de codificación de audio y video disponibles en el hardware por medio de **CFG_Download**
5. Determinar la entrada de video que se va a emplear y el formato en el que se encuentra la señal en la entrada por medio de **CFG_VideoPreview**
6. Abrir la entrada por medio de **RTE_OpenSession**
7. Detectar que entrada tiene una señal por medio de **CFG_DetectVideoInput** y comparar contra la deseada en los parámetros iniciales
8. Habilitar la interfaz de captura compatible con ISMA por medio de **RTE_EnableISMASession**
9. Crear un FileSink a partir de la interface ISMA obtenida en el paso anterior por medio del método **CreateNewFileSink**
10. Crear un *NetworkSink* a partir de la interfaz ISMA obtenida en el paso anterior por medio del método **CreateNewNetworkSink**
11. Establecer la resolución de video a la cual se capturará por medio de **CFG_EncodingResolution**

12. Establecer la ganancia de audio de ambos canales de audio por medio de **CFG_AudioGainLeft y CFG_AudioGainRight**
13. Determinar la entrada de audio y el muestreo a utilizar mediante la llamada **CFG_AudioPreview**
14. Inicializar la estructura *OUTPUTINFO*
15. Inicializar la estructura *NOTIFYINFO* para determinar la instancia que recibirá las notificaciones de la entrada
16. Inicializar la estructra del tipo **LocalStreamSpecificParams** con los parámetros con los que se desea realizar el flujo de datos del video y llamar **CFG_VideoBitrateConfigEx**
17. **CFG_VideoPesLayer** tanto para el sink de video como el de audio
18. **CFG_AudioPesLayer** tanto para el sink de video como el de audio
19. Inicializar el FileSink por medio de la estructura ***IFileSinkEx***
20. Inicializar el NetworkSink por medio de la estructura ***IStreamNetworkParameters***
21. Configurar el overlay de texto con el método **CFG_SetTextVideoOverlay**
22. Llamar el método **RTE_InitSession.**

Una vez que logré configurar e incializar correctamente el objeto CRte el capturar y transmitir el video de la entrada fue una tarea sencilla, ya que solamente requería de la llamada a los métodos **RTE_StartSession** y **RTE_StopSession** respectivamente:

Código para finalizar una sesión

Para finalizar y cerrar correctamente la entrada programáticamente, primero tuve que detener la sesión de captura, después la de transmisión, para poder ejecutar lo siguiente:

```
void CloseInput(void)
{
   //Cierra la sesión
   if(RTE_Handle)
   {
        RTE_CloseSession(RTE_Handle);
   }

   //Cierra el handle de configuración
   if(RTE_Handle && RTE_ConfigHandle)
   {
        RTE_CloseConfig(RTE_Handle, RTE_ConfigHandle);
        RTE_ConfigHandle = NULL;
   }
   //Cierra el handle del objeto
   if(RTE_Handle)
   {
        RTE_Close(RTE_Handle);
        RTE_Handle = NULL;
   }
}
```

donde:
RTE_Handle es del tipo HRTE
RTE_ConfigHandle es del tipo HCFG

3.3.5 Capas de abstracción

En las siguientes figuras se muestran las capas de abstracción de la manera en la que utilicé las herramientas para implementar el sistema:

Servidor

Figura 3.15 - Capa de abstracción en el servidor

Cliente

Figura 3.16 - Capa de abstracción en el cliente

Es importante notar que gran parte del software que desarrollé utiliza MFC para realizar la mayoría de sus tareas, sin embargo también utilicé diversas bibliotecas que ofrece Microsoft como lo son WinSock, para el manejo de sockets, WinAPI, principalmente para consultas al sistema operativo, consultas al sistema de archivos, lectura de registros de sistema, etc.

3.4 Análisis Previo al Desarrollo

Antes de comenzar a describir el proceso de desarrollo del software, es indispensable incluir en el escrito la forma en la que decidí que método de intercomunicación de procesos era el más indicado para la interacción de los módulos, previamente definida en el diseño de acuerdo a la figura que muestra la interacción de los módulos.

En esta sección también describiré cómo realicé el análisis de la búsqueda de los archivos multimedia capturados en base a la fecha, hora y canal de una forma conveniente para el sistema.

3.4.1 Análisis de la Comunicación Interprocesos

Para separar el sistema en diferentes programas dentro de Windows, fue necesario que investigara los mecanismos que ofrece el sistema operativo para comunicar a los procesos entre sí.

Windows ofrece diversos mecanismos para la Comunicación Interprocesos (IPC) de acuerdo a la documentación albergada en la Red de Desarrolladores de Microsoft (MSDN) ubicada en la Internet y estos son los siguientes:

1. Clipboard
2. COM (Component Object Model)
3. Data Copy
4. DDE
5. File Mapping
6. Mailslots
7. Pipes
8. RPC
9. Windows Sockets

De acuerdo a Microsoft, es importante mi elección estuviera basada en mis respuestas al cuestionario que se muestra a continuación, para así poder seleccionar el método IPC que mejor se adapte a las necesidades del proyecto:

- ¿El programa debe ser capaz de comunicarse con otros programas ubicados en el mismo ordenador o debe ser capaz de comunicarse con otros procesos ubicados dentro de una red de computadoras?
- ¿El programa debe ser capaz de comunicarse con otros programas que son ejecutados en otros ordenadores que corren bajo distintos sistemas operativos como Linux, MacOS, etc.?
- ¿El usuario final debe ser capaz de elegir los otros programas con las que se comunicará el programa o el programa sabe implícitamente los programas con los que debe de comunicarse?
- ¿El programa debe ser capaz de comunicarse con otros programas de forma general usando la mecánica de copiar-y-pegar datos del portapapeles o su comunicación con el resto de los programas está delimitado a una interacción muy específica?
- ¿El desempeño es un factor importante dentro del programa?
- ¿El programa será un programa con GUI o simplemente un programa de consola?

En base a lo establecido en el diseño, me resultó lógico concluir que el programa con mayor jerarquía dentro del proyecto era el módulo *Fission Monitor Server*, ya que éste tiene como tarea la acción principal de capturar las señales de A/V. Ya que todos los módulos trabajan alrededor de éste, y tienen que adaptarse a la forma de comunicación del servidor.

Tomando en cuenta lo anterior como premisa y siguiendo el cuestionario, reflexioné que el servidor tenía que ser capaz de comunicarse con programas desarrollados por terceros o con los otros módulos de Fission en un futuro. El origen de esta idea nace del hecho de que las soluciones implementadas dentro las televisoras son multidisciplinarias e implementadas por diferentes compañías.

Mi experiencia profesional dictó que era altamente probable que los programas externos fueran ejecutados en distintos sistemas operativos y en ordenadores distintos al que alberga el sistema que desarrollé. También consideré que era altamente probable que las aplicaciones externas solicitaran comunicarse con el servicio de captura, con lo que fue razón suficiente para que desarrollara un protocolo propietario de comunicación hacia el módulo *Server*.

En el caso de los módulos *Server*, *Upkeeper* y *Exporter* no es necesaria una GUI ya que el módulo *Client* brindará la GUI del sistema en general.

En base a las respuestas anteriores y tomando en cuenta las propiedades de cada uno de los métodos proporcionados por Microsoft, pude concluir lo siguiente para cada método de intercomunicación que ésta ofrece:

El ***Clipboard*** (o Portapapeles en español) funciona como un depósito central para compartir información entre todos los programas corriendo sobre el sistema operativo, esta memoria es la que se utiliza cuando el usuario realiza la acción de *Cortar y Pegar,* por lo tanto consideré que no es un medio confiable, ya que diversas aplicaciones tienen acceso a ese tipo de memoria compartida abriendo una posibilidad al error y pérdida de mensajes, ya que alguna otra aplicación podría modificar o eliminar un mensaje de la pila.

Por otro lado, ***COM*** (Modelo de Objetos Componentes) fue creado con la finalidad de compartir información entre editores de documentos por medio del uso de la tecnología OLE. El uso de la tecnología OLE no es trivial y no cumple con las necesidades de intercomunicación del sistema que desarrollé debido a que sólo buscaba compartir mensajes de texto y no mensajes tan complejos como los que ofrece este método.

Data Copy lo tomé en consideración, dado que utiliza el mensaje nativo de Windows WM_COPYDATA y un formato en el mensaje que tanto el programa que envía como el que recibe estén en mutuo acuerdo, éste método se utiliza para enviar información de una manera rápida entre aplicaciones. Por lo tanto se utilizaría entre las aplicaciones que se ubiquen en la misma parte del servicio, es decir, para la comunicación entre el módulo *Upkeeper* y el *Server*; y la comunicación entre el módulo *Client* y *Exporter*.

El protocolo **DDE** es una capa establecida sobre el mecanismo del porta-papeles, por lo que lo descarté inmediatamente.

Un mecanismo interesante es el llamado ***File Mapping*** ya que éste permite a un proceso el tratar el contenido de un archivo como si fuera un bloque de memoria y de esta forma poder utilizar apuntadores. Sin embargo, esto limita a la comunicación que solamente se encuentren dentro del mismo ordenador y se encuentra a la par del mecanismo *Data Copy*. En comparativa, *File Mapping* necesitaba un desarrollo más complejo por mi parte para obtener el mismo resultado que *Data Copy,* él cuál solo

requería de mi parte programar una atención a un mensaje más de Windows, lo cual es relativamente trivial de programar.

Mailslot funciona como un servicio de correo donde el mensaje se retiene en el buzón de entrada hasta que haya sido leído por el proceso receptor o servidor, los mensajes pueden ser enviados por medio de un broadcast o dirigidos desde un proceso cliente, sin embargo el tamaño del mensaje está limitado a 400 bytes, en caso de que sea un broadcast, o al tamaño que haya sido inicializado el buzón del receptor. Razoné que la limitante del mensaje podría ser una limitante a futuro cuando el software requiere de una expansión de capacidades, por ese motivo, lo descarté como una opción.

RPC (o Llamada Remota de Procesos de sus siglas en inglés) es un mecanismo que habilita a los programas el llamar funciones remotamente, permite que la comunicación sea tan fácil como el hacer el llamado a un método o función. Este puede funcionar en el mismo ordenador o en uno distinto dentro de una red local. Pero para usar este método tenía que implementar una definición del lenguaje, mediante el *Microsoft Interface Definition Language* o MIDL, lo cual limitaba a la comunicación entre procesos ubicados en el mismo servidor y seguía resultando más atractivo el método Data Copy por su sencillez. Otra desventaja que encontré en este mecanismo fue que el lenguaje que hubiera definido se creaba al momento de compilar el programa, obligándome a compilar todos los módulos si este fuera el método de comunicación entre ellos, irrumpiendo con la independencia que buscaba al momento de actualizar los módulos para la corrección de errores o mejora de procesos.

Otro mecanismo utilizado ampliamente por los programadores son los ***Pipes***. Un *Pipe* es un medio de comunicación entre procesos relacionados para transferir información entre ellos por medio del uso de memoria compartida. El proceso que crea el pipe es el servidor de pipes y el proceso que se conecta a dicho pipe es el cliente, por lo tanto me resultaba atractivo este método, sin embargo nadie dentro del personal de Fission contaba con la experiencia previa en este método e implicaba un coste en tiempo durante el desarrollo debido al aprendizaje que este método implicaba.

Finalmente, los **Windows Sockets** es una interfaz independiente del protocolo que puede comunicarse con otra implementación de sockets de otros sistemas. Los sockets en Windows pueden ser usados en base las mismas funciones estandarizadas de E/S que con las que se manipula un archivo.

La compatibilidad con sistemas externos que el último método implica, me empujó a decidir que era la mejor opción como método de comunicación para el proyecto. Este método provee una comunicación rápida C/S y también permite la comunicación con un programa ubicado en un ordenador remoto, lo cual me permitiría desarrollar una solución escalable. Una vez que tomé la decisión, encontré que existen dos tipos de sockets principalmente: UDP y TCP. La decisión sobre que tipo elegir dependía del tipo de tarea que ejecutaría el módulo y los datos que requiera comunicar.

3.4.2 Sockets

TCP

La comunicación por medio de TCP (Protocolo de Control de Transmisión, de sus siglas en inglés) es el método más usado en las redes de computadoras y sobre todo en la Internet. Esto se debe a que el TCP tiene corrección de errores, por lo tanto se tiene una garantía de que los paquetes de datos se entregaron correctamente con base en el método llamado *Control de Flujo*. El *Control de Flujo* determina cuando los datos requieren ser renviados y detiene el flujo de datos hasta que el paquete previamente perdido es transferido. Este método funciona muy bien ya que en la comunicación en

redes es muy probable que sucedan colisiones de paquetes. Cuando una colisión ocurre, el cliente remoto puede volver a pedir el paquete perdido hasta que sea igual al original enviado por el servidor.

UDP

El protocolo UDP (Protocolo de Datagrama de Usuario, de sus siglas en inglés) es también utilizado con popularidad en la Internet, sin embargo, nunca se usa en el envío de información importante como páginas web, información de bases de datos, etc.; más bien, su uso principalmente radica en el streaming. La ventaja que aprecié en este protocolo contra el TCP era la velocidad que ofrece, ya que no tiene un control de flujo o corrección de errores. Sin embargo, esto implica que el mensaje pueda ser perdido durante la comunicación. El propósito de este protocolo es, principalmente, el de enviar un flujo constante de información (como audio y video digital) el cuál el cliente remoto sabrá descifrar y determinar si hubo algún error o no en la transmisión.

En el caso en particular del software que desarrollé, el protocolo UDP jugaba un papel muy importante, ya que este protocolo es la base del protocolo RTP (Protocolo de Transporte en tiempo Real, de sus siglas en inglés) y es utilizado por el RTE SDK de Optibase para visualizar la señal de audio y video de cada una de las entradas de la tarjeta Optibase que utilicé.

Para comprender la razón de por que el UDP es más rápido que el protocolo TCP, es necesario que explique el Modelo de Referencia OSI (Interconexión de Sistemas Abierto, de sus siglas en inglés) desarrollado por la ISO (Organización Internacional de Estándares, de sus siglas en inglés). Dicho modelo establece una referencia común sobre la cuál se desarrollan los protocolos de comunicación.

Modelo OSI

Dicho modelo consiste de siete capas que define la funcionalidad de los protocolos de comunicación. Cada capa representa una función ejecutada cuando los datos son transferidos entre aplicaciones cooperativas a través de una red. En el siguiente diagrama se puede apreciar como los protocolos basados en el modelo OSI parecen una pila de bloques apilados uno sobre otro, por esta razón a la estructura de los paquetes se les conoce como pila o pila de protocolo.

Figura 3.17 - Capas Modelo OSI

Una capa no define solamente a un protocolo en particular, define la función que debe llevar a cabo durante la comunicación los de datos sin importar el protocolo. Por ejemplo, un protocolo de transferencia de archivos y un protocolo de correo electrónico proveen un servicio al usuario, por lo tanto ambos protocolos se ubican dentro de la *Capa de Aplicación*.

En mi caso de estudio, es importante considerar que los protocolos TCP y UDP se encuentran implementados en las capas superiores a la *Capa de Transporte* en una aplicación práctica, ya que en las capas inferiores está el protocolo IP (Protocolo de Internet, de sus siglas en inglés) el cuál ya se encuentra implementado a nivel del controlador de una tarjeta de red en el sistema operativo o en el mejor de los casos, ya está implementado en el hardware dedicado a la comunicación por red.

Al momento de enviar información, ya sea por TCP o UDP, ésta se divide en paquetes y se encapsula. La encapsulación de la información se genera mediante la adición de más datos de control que se denominan encabezados. Por cada capa descrita en el modelo OSI, se va agregando un encabezado a la información a ser enviada hasta llegar a la capa física de transmisión, una vez que el cliente remoto recibe la información, éste desencapsula el paquete conforme el protocolo va escalando por sus capas, hasta quitar todos los encabezados y así obtener los datos originales.

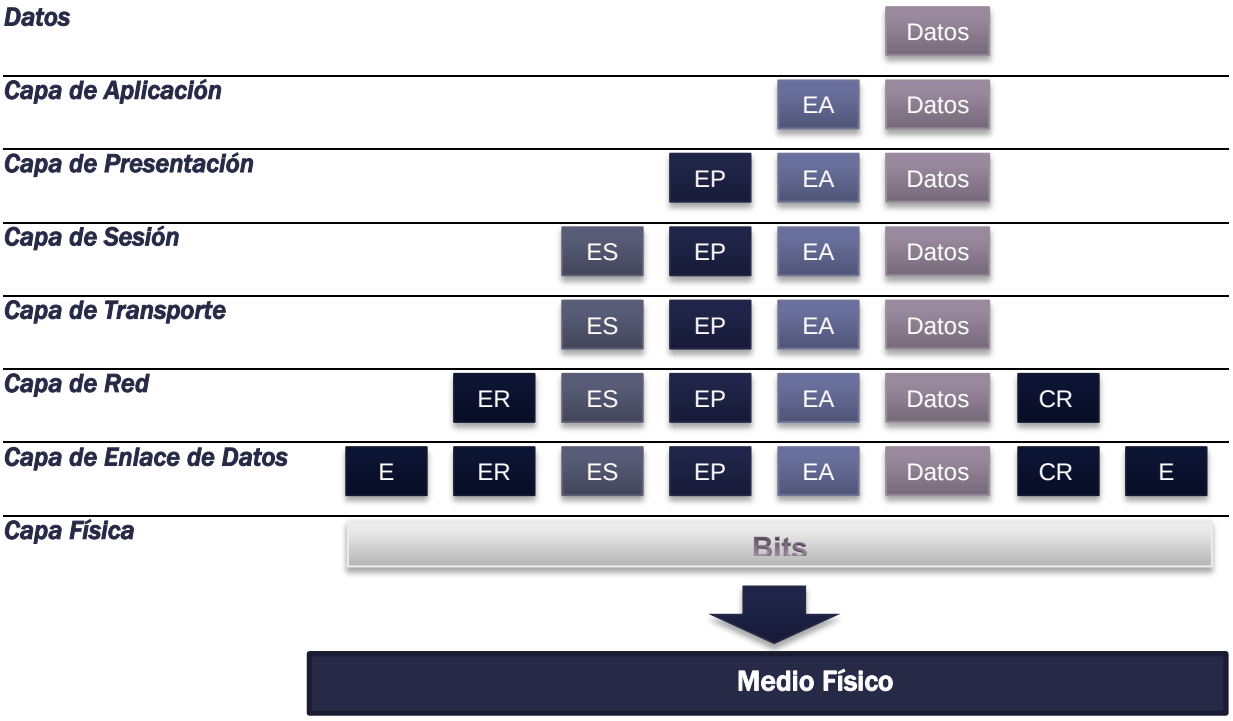


Figura 3.18 - Encapsulamiento de datos en el modelo OSI

TCP vs UDP

Para realizar una comparación práctica de ambos protocolos las siete capas del modelo OSI se pueden contraer en solamente cuatro capas de acuerdo a Hunt, donde los datos fluyen de la misma manera que en el modelo OSI de arriba hacia abajo y viceversa como se muestra en la siguiente figura:

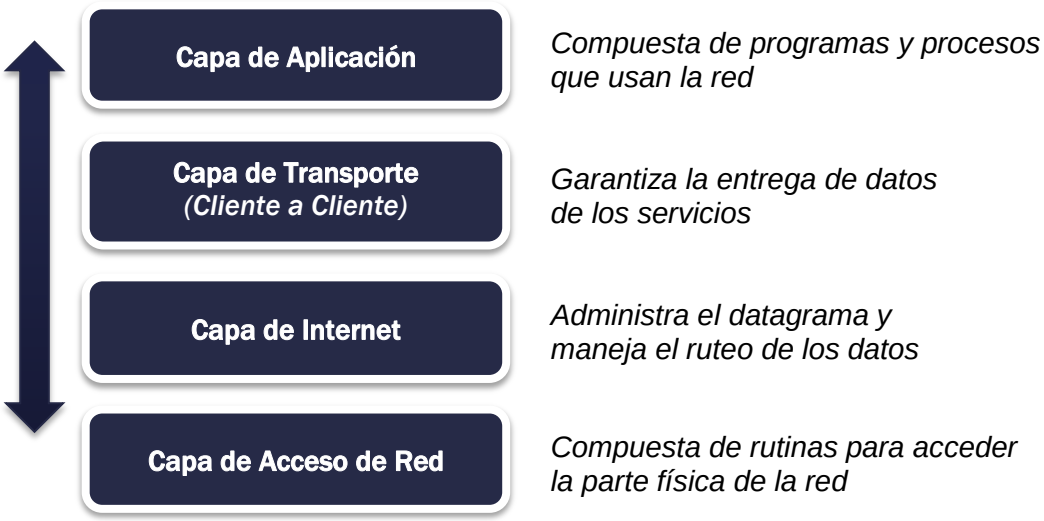


Figura 3.19 - Abstracción de capas para sockets TCP y UDP

Ambos protocolos son similares entre sí, aunque los términos que se utilizan en cada capa, de este modelo reducido, son distintos. En la capa de aplicación, en TCP se refieren a los datos como stream o *flujo*, mientras que en UDP se utiliza el término mensaje. En la capa de transporte: en TCP se refieren a los datos como segmento y en UDP como paquete. En la capa de Internet: a los bloques datos se les denomina datagramas en ambos protocolos, como se muestra en la siguiente figura:

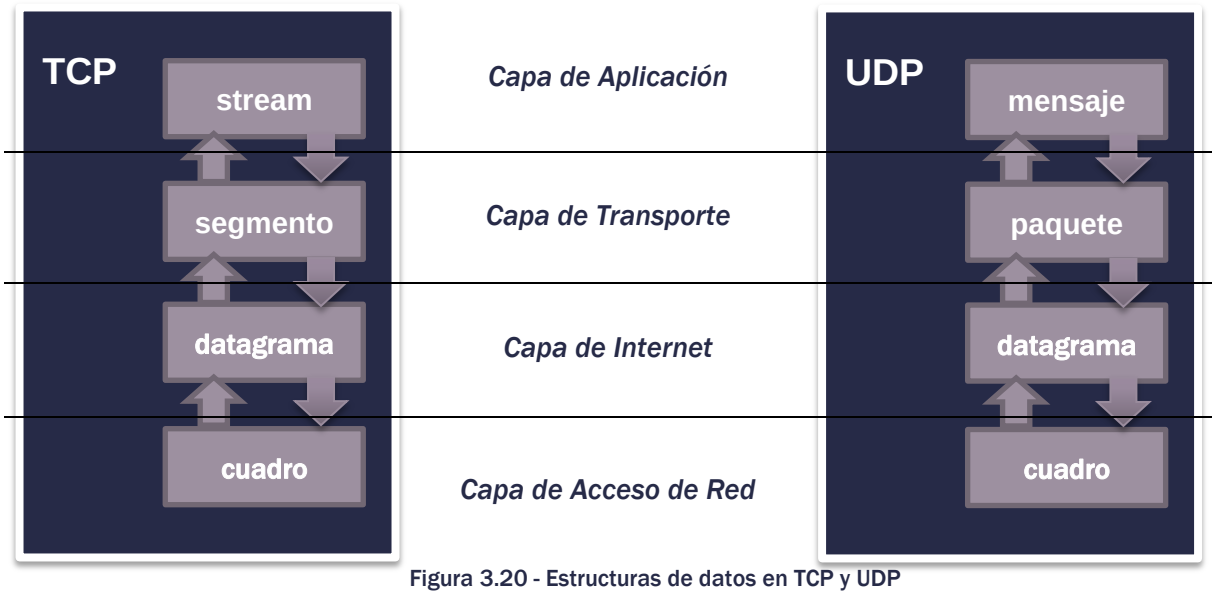


Figura 3.20 - Estructuras de datos en TCP y UDP

A pesar de que en la arquitectura no se muestra una diferencia significativa, ésta se deja notar en el encabezado que se realiza sobre los datos a transmitir como se nota con mayor detalle en las siguientes figuras.

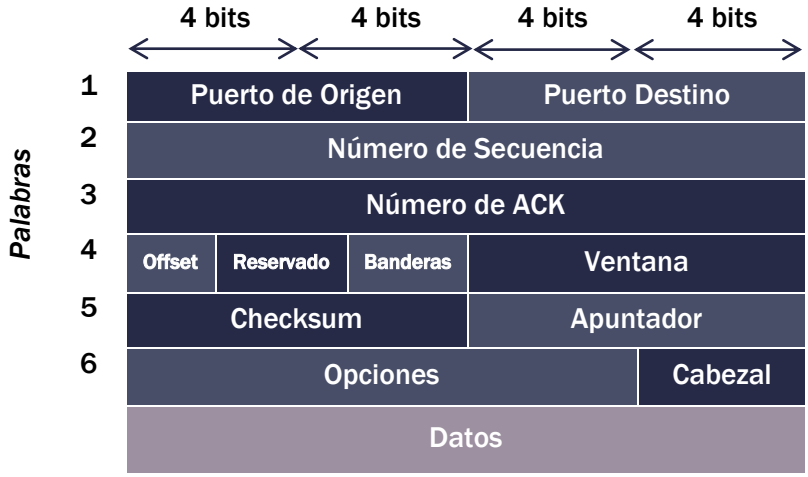


Figura 3.21 - Datagrama TCP

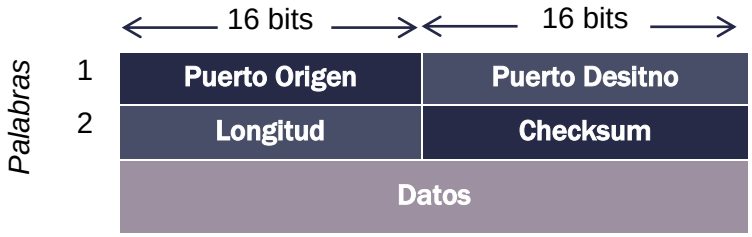


Figura 3.22 - Datagrama UDP

Cabe recordar que el protocolo TCP es utilizado como un medio confiable para la transmisión de datos con detección de errores, mientras que UDP provee un servicio de envió de datagramas con poco encabezado y sin conexión.

Sin embargo, cuando comencé a desarrollar el software, contaba solamente con una implementación previa, de mi autoría, de los socket UDP. Considerando que el sistema, en su etapa inicial, contaría con todos los módulos dentro del mismo ordenador, concluí que era viable programar, en la primera entrega, toda la comunicación por medio de sockets UDP. Esta decisión la tomé tomando en cuenta que, en la actualidad, si dos programas se comunican por medio de sockets UDP dentro del mismo ordenador, la tecnología ha madurado lo suficiente para que la pérdida de paquetes sea nula.

No obstante, considero que en un futuro se deben de implementar la comunicación mediante sockets TCP para la transmisión de comandos y peticiones del sistema y limitar la utilización de los sockets UDP para el streaming y envío de datos continuo como los contadores de grabación que se tienen en cada entrada, para así obtener mayor escalabilidad al software.

3.4.3 Búsqueda de Archivos

Como se mencionó en los requerimientos, el sistema debe de ser capaz de mostrar al usuario el momento exacto del video capturado basado en una fecha, hora y canal específicos. Lo que me obligó a diseñar e implementar un procedimiento de búsqueda dentro del registro de captura.

Con el término *registro de captura,* me refiero a todos los archivos grabados mediante el hardware de captura de video. Cada uno de estos archivos lo denominé *segmento del registro*. La duración del segmento está definida por el usuario y puede cambiar dinámicamente durante la grabación. Por lo que se tenía que tomar en cuenta que los segmentos dentro del registro pueden llegar a tener una duración variable.

Tomando en cuenta que los *segmentos del registro* son archivos dentro de un directorio que son administrados por el sistema de archivos del sistema operativo. Se me ocurrió que los archivos de video estuvieran nombrados, de tal forma, que al ordenarlos alfabéticamente también se ordenaran de manera cronológica. Así podría aprovechar la llamada a sistema que obtiene un listado ordenado alfabéticamente de los archivos ubicados dentro del directorio asignado a almacenar los archivos capturados. Logrando de esta forma, simplificar la búsqueda a un proceso de búsqueda lineal dentro del listado resultante de la llamada al sistema.

Por lo tanto, decidí crear una nomenclatura de los archivos que tome ventaja de ello.

Nomenclatura de los archivos capturados

El nombre debe de describir la captura del archivo, por lo que lo diseñé para que contenga los siguientes valores: la entrada desde la que se capturó, el nombre del canal para el cuál se capturó, la fecha y hora de inicio de la grabación.

El orden de los campos es indistinto, excepto el orden en el que se coloque la fecha con respecto a la hora. La fecha siempre debe de anteceder a la hora y estar descrita en año-mes-día. Esto tiene origen en el ordenamiento alfabético que se realiza de izquierda a derecha con respecto al orden de los caracteres contenidos en el nombre.

Descrito lo anterior, a continuación se listan las reglas que dicté para formatear el nombre de los archivos capturados:

1. El nombre del archivo debe estar formado por cuatro campos alfanuméricos:
    1. el número de entrada por la cual fue capturado el archivo
    2. el nombre del canal
    3. la fecha en la que inició la captura del archivo
    4. la hora en la que inició la captura del archivo
2. Los campos deben estar separados por un caracter de guion bajo "_", consecuentemente, el uso de este caracter está prohibido dentro del valor de cualquiera de los cuatro campos
3. El primer campo se debe de nombrar como "In**#**" donde # debe ser remplazado por un dígito, dígito que corresponde a la entrada por la cual fueron capturados el video y audio. Este número es único por entrada A/V del ordenador y su valor se encuentra en el siguiente rango: [1,6]
4. El segundo campo debe contener el nombre del canal. El nombre está formado por una cadena de caracteres que no puede utilizar el guión bajo o cualquiera de los siguientes caracteres: **/, \, ?, %, *, :, |, ", <, >**. debido a que dentro del NTFS está prohibido el uso de esos caracteres en el nombramiento de un archivo. El máximo número de caracteres permitidos para el nombre del canal es de 231, ya que el número máximo de caracteres permitidos en el nombre de un archivo en NTFS es 255
5. El tercer campo consta de 8 caracteres. Los primeros cuatro describen el año, los siguientes dos el mes y los últimos dos el día del mes de cuando comienza la fecha correspondiente a la captura del archivo, es decir, **aaaammdd**
6. El cuarto campo está formado por 6 caracteres. Los primeros 2 corresponden a la hora en formato de 24 horas, los siguientes dos a los minutos y los últimos dos a los segundos cuando el archivo comenzó a ser capturado. Es decir: **HHMMSS**
7. Finalmente el archivo debe contener el sufijo "**.mp4**" ya que esta es la extensión correspondiente al formato en el cual es capturado el archivo por la tarjeta Optibase MM400. En caso de ser capturado el archivo en algún otro formato esta extensión tendrá que sufrir un cambio al igual que el número de caracteres máximo para el nombre del canal en el punto 4.

Por lo tanto, el archivo quedaría nombrado de la siguiente forma:

In*#_$_aaaammdd_HHMMSS***.mp4**

Donde:
El caracter **#** representa el número de entrada.
El caracter **$** representa el nombre del canal
El grupo **aaaa** representa el año de inicio de captura del archivo.
El grupo **mm** representa el mes de inicio de captura del archivo.
El grupo **dd** representa el día de inicio de captura del archivo.
El grupo **HH** representa la hora de inicio de captura del archivo.
El grupo **MM** representa los minutos de inicio de captura del archivo.
El grupo **SS** representa los segundos cuando dio inicio de captura del archivo.

Para ilustrar el uso de estas reglas, coloco el siguiente ejemplo:

In3_TNT_20101216_154609.mp4

El nombre de este archivo indica que el archivo fue capturado por la **tercer** entrada del sistema, donde el canal posee el nombre **TNT** y el tiempo de inicio de captura de la multimedia son las **15:46:09** horas del **16 de diciembre del 2010**.

Filtrado de Archivos

Una vez que logré obtener un registro de captura poblado con archivos nombrados en base a las reglas que establecí, pude obtener, fácilmente dentro del programa, el listado de esos archivos ordenados alfabéticamente mediante llamadas al sistema operativo, filtrando el resultado por medio de una expresión regular como la siguiente:

In/d_$_\d\d\d\d\d\d_\d\d\d\d\d.mp4

Donde **$** es el nombre del canal que deseo buscar. Al obtener un listado filtrado de los archivos, pude delimitar la lista a contener solamente los archivos de un canal en específico. Se le recuerda al lector, que los archivos pueden estar ubicados en un directorio de captura y en un directorio de almacenamiento a largo plazo, era necesario que implementara este proceso de búsqueda restringida, ya que los archivos correspondientes a un canal se pueden encontrar en más de un directorio.

Una vez que obtuve las listas ordenadas de los archivos correspondientes a un canal, resultó indispensable que implementara un proceso que mezcle ambas listas y me diera como resultado una lista igualmente ordenada que las originales. Razón por la que utilicé el algoritmo de Ordenamiento por Mezcla.

Ordenamiento por mezcla

Para realizar una búsqueda global de los archivos correspondientes a un canal dentro del registro de captura, era necesario que converja las listas de los archivos que se encuentran ubicados en los directorios de captura y la de los archivos en el almacenamiento masivo.

La descripción del algoritmo la haré por medio de un ejemplo. Primero se considera que se tienen dos listas ordenadas: la lista **A**, para los archivos ubicados en el directorio de captura y la lista **B**, para los archivos ubicados en el almacenamiento masivo. Las listas están conformadas por los elementos *A.1*, *A.2*, ..., *A.x* y *B.1*, *B.2*, ..., *B.y* respectivamente, donde *x* y *y* son el tamaño respectivo de cada lista. Para efectos didácticos se define *x=3* y *y=2*.

Figura 3.23 - Listas Ordenamiento por Mezcla

Inmediatamente se crea una tercer lista, C, de tamaño z = *x+y = 5,* la cual será el resultado del ordenamiento. Consecutivamente, se comparan los primeros elementos de ambas listas y se agrega el elemento más pequeño de la comparación en la primera posición disponible de la lista C, en este caso A.1 < B.1, el elemento A.1 se copia en C.1.

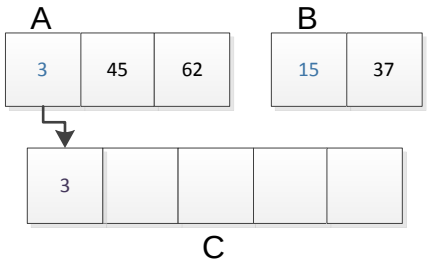


Figura 3.24 - Ordenamiento por Mezcla paso 1

Una vez añadido el primer elemento en la lista C, se mueve la posición del elemento a comparar en la lista A hacia el elemento sucesivo, A.2 y se vuelven a comparar los elementos. En esta ocasión B.1 < A.2, por lo tanto se coloca B.1 en C.2.

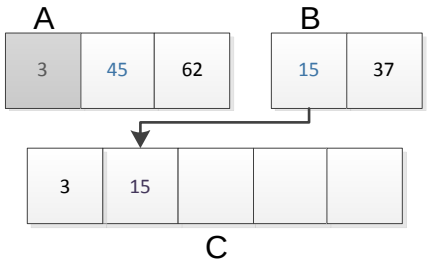


Figura 3.25 - Ordenamiento por Mezcla paso 2

Se mueve la posición de comparación de la lista B a B.2. Esta vez, dado que B.2<A.2 se coloca B.2 en C.3.

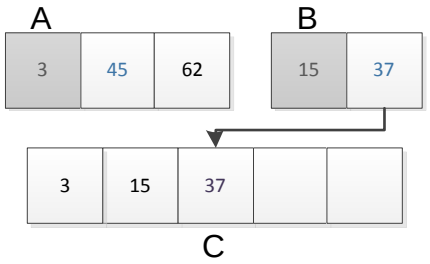


Figura 3.26 - Ordenamiento por Mezcla paso 2

Finalmente dado que se terminaron los elementos de la lista B, se colocan los elementos restantes de la lista A en forma secuencial en la lista C.

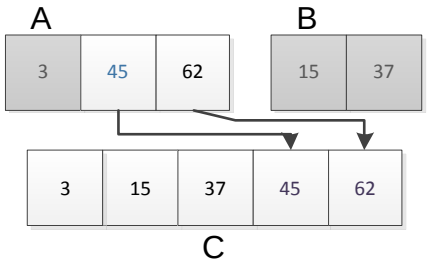


Figura 3.27 - Ordenamiento por Mezcla paso final

Cuando implementé este algoritmo dentro del sistema, contemplé que los valores a comparar, entre los nombres de archivos, tenían que ser la fecha y hora de inicio de grabación, para así tener como resultado el listado de elementos únicos, ordenados cronológicamente de los archivos correspondientes a un canal.

Concretando la descripción del proceso de búsqueda que implementé, una vez que obtuve la lista ordenada de los archivos de un canal, realicé una búsqueda lineal en dicha lista basada en la fecha y hora de inicio de grabación, información contenida en el nombre del archivo. Como resultado de la búsqueda lineal obtuve un archivo que **posiblemente** contiene el video que se busca. Para confirmar si este es el archivo que contiene la multimedia esperada, fue necesario que abriera el archivo para obtener la duración del mismo y así poder acreditar que es el archivo de video correspondiente al a los parámetros iniciales de la búsqueda realizada sobre el registro de captura.

Este último proceso lo describo con mayor detalle en el siguiente diagrama de flujo:

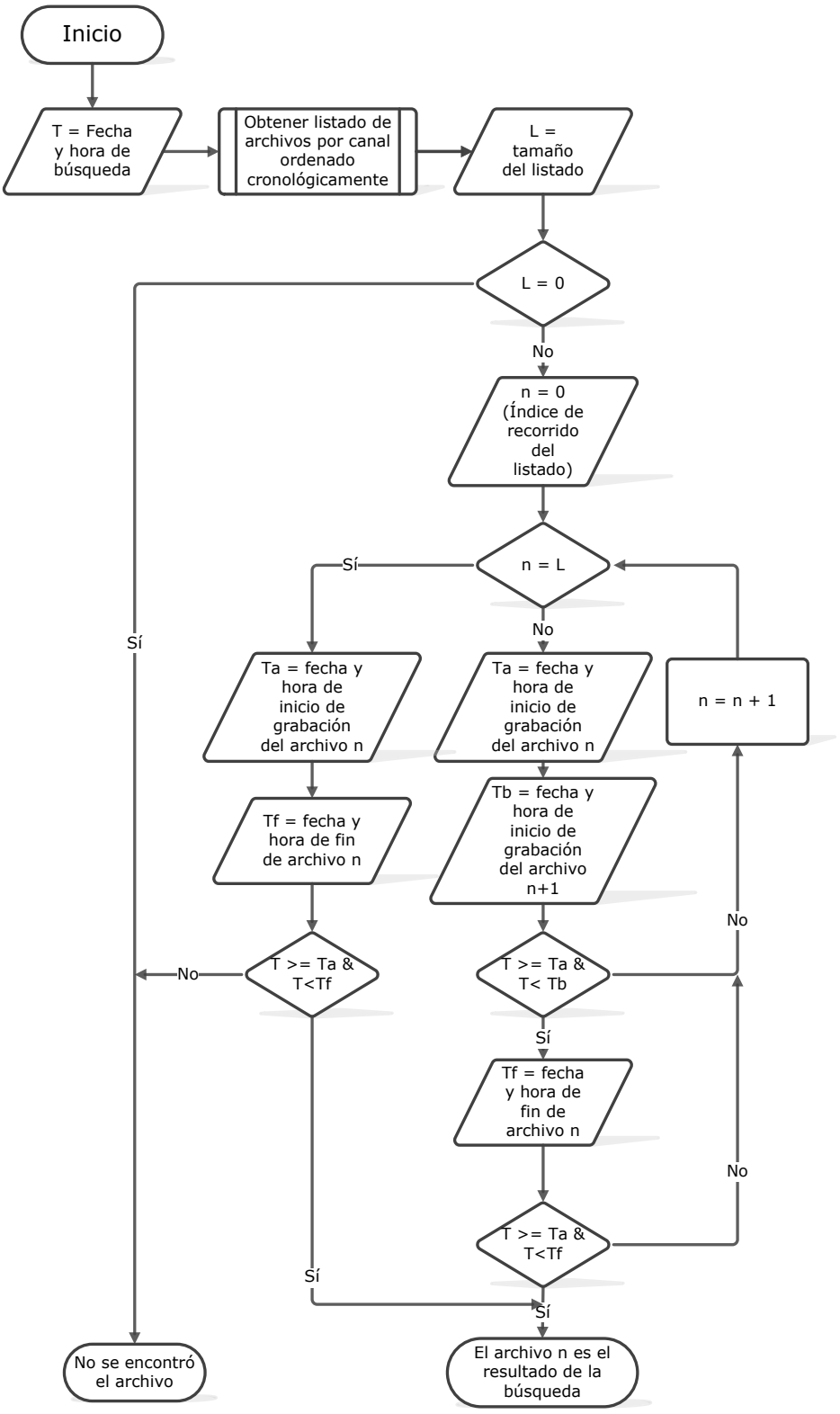


Figura 3.28 - Diagrama de flujo búsqueda de archivo

3.5 Desarrollo

3.5.1 Implementación y Distribución de Versiones

Antes de comenzar a describir los puntos importantes relacionados al desarrollo del sistema, es importante dar a conocer la manera en que se asignan las versiones del software que liberaba a los clientes, para así tener un control adecuado durante el desarrollo dinámico bajo el que trabajé. La nomenclatura de las versiones de los programas está basada en 4 campos numéricos separados por un punto, ya que así lo ofrece la herramienta Visual Studio, por ejemplo:

`1.0.3.45`

Dichos campos incrementan a razón de 1 basado en el avance que iba obteniendo sobre el producto.

El primer campo indica el número de la versión del producto. Éste campo no lo podía cambiar, a menos de que la compañía me solicite su cambio. Debido a que este campo es carta de presentación ante los clientes.

El segundo número marca una subversión del producto que consideraba con funcionalidad estable. Este campo engloba todos los cambios que fueron requeridos por diversos clientes y muestra mejoras notables sobre la subversión anterior a este nivel.

Recordando que la forma de desarrollo, dentro de Fission Software, está basada en el modelo Iterativo Incremental, en el tercer campo se lleva el registro del incremento de la funcionalidad del módulo. Cada iteración indica que existe una funcionalidad dentro del sistema que no se tenía con anterioridad.

En el último campo se registran las correcciones de errores de programación, incongruencias lógicas o correcciones a incomprensiones con el cliente con respecto a las funcionalidades solicitadas por los clientes. Este campo dicta explícitamente que hubo una mejora con respecto a la funcionalidad descrita en el tercer campo.

Finalmente, es importante tomar en cuenta que al momento de que incrementaba cualquier campo, los campos siguientes los reiniciaba al valor de cero.

Una vez que compilaba una versión lista a liberar con los clientes, con su número de versión correspondiente, era necesario que distribuyera los ejecutables junto con unas notas de lanzamiento contenidas en un archivo de texto con extensión .txt, llamado *ReleaseNotes.txt*. Este archivo es una bitácora útil para el departamento de operaciones, ya que por medio de éste, el personal de Operaciones podía saber que avances tuvo la aplicación o el sistema en general. Dichas notas llevan un formato como el siguiente:

Fecha de lanzamiento (YYYY-MM-DD)
[Listado de ejecutables o DLL´s modificadas con número de versión]
Descripción:
-[Listado de modificaciones realizadas]

Un ejemplo de una entrada a las notas de lanzamiento correspondiente a una versión lista para instalarse se vería como:

```
2009-11-13
This bundle updates the following versions:
  OAClient.exe               version 0.9.4.10
  FissionCore.dll            version 1.0.1.1

Description:
      OAClient
      - Fixed bug where spontaneous errors occurred while searching for
      an existing date and time (Added a log named MP4Seeker.log for
      further analysis on this bug).
      - Fixed Now Time Label in the Record Viewer which didn't update
      correctly.
```

Conforme iba liberando las versiones, las entradas más recientes la iba colocando al inicio de las notas de lanzamiento

Una vez que entregaba una versión al área de operaciones, dentro de la compañía es un requisito que diera de alta las modificaciones realizadas en el código, para dicho lanzamiento, en el repositorio de subversiones. Las notas que colocaba en el sistema de subversión son notas que deben ir dirigidas al equipo de desarrollo, además de la documentación incluida en el código por medio de comentarios insertados en el código.

3.5.3 Fission Monitor Server

Como mencioné en el capítulo de Diseño, el módulo Server está encargado de ofrecer el servicio de captura de hasta 6 entradas. Esta aplicación la programé como un servicio que responde a comandos recibidos por sockets UDP, que posteriormente son procesados por un parser, que también programé con la finalidad de verificar la sintaxis de los comandos contra el protocolo bajo el que está implementado.

El parser, después de procesar el comando, entrega un mensaje a un intérprete, para determinar si el mensaje es semánticamente correcto, por lo tanto también tuve que implementar este componente. En caso de que el mensaje no fuera correcto, envía un mensaje de error al cliente como respuesta y en caso de que contenga el formato correcto, éste es traducido a un comando comprensible al proceso ejecutor, proceso que lleva a cabo la acción solicitada por el usuario externo. Las diferentes tareas que el servidor puede realizar las agrupé en dos categorías principales:

- Configuración del sistema
- Manipulación de la grabación

El proceso que configura el módulo almacena en el registro del sistema operativo los valores necesarios para configurar el mantenimiento de archivos, así como la manera en la que se debe de realizar la grabación. Algunas modificaciones a la forma de grabar se pueden realizar mientras el módulo está grabando, pero otras requieren que se reinicie la grabación para tomar efecto, ya que el hardware así lo requiere.

La manipulación de la grabación las realiza el módulo directamente con un proceso que implementé llamado *Administrador de Hardware*. Éste es un componente

que controla el hardware de grabación, ejecutando un comando a la vez. En caso de que reciba múltiples mensajes para la manipulación de las tarjetas de grabación, en este punto se encolan conforme van llegando para su ejecución.

Por otro lado el módulo de mantenimiento Fission Monitor Upkeeper se ejecuta cíclicamente en un tiempo que predeterminé de 10 minutos, para dar mantenimiento a los directorios de grabación y almacenamiento masivo.

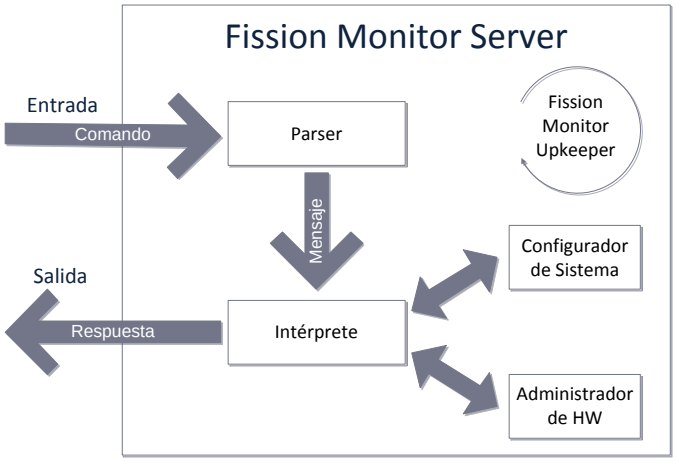


Figura 3.29 - Diagrama Fission Monitor Server

Entrada A/V como máquina de estados

Antes de describir cada uno de los componentes del servidor, es importante recapitular que cada entrada de A/V la consideré como una máquina de estados finita no determinística, para facilitar su diseño lógico y por lo tanto su programación.

El autómata tiene como entrada un mensaje que formatea el *Intérprete* del módulo, al cual se denomina comando y tiene como salida un mensaje de regreso, que es procesado por el *Intérprete* y conforme a éste último envía la respuesta correspondiente al cliente.

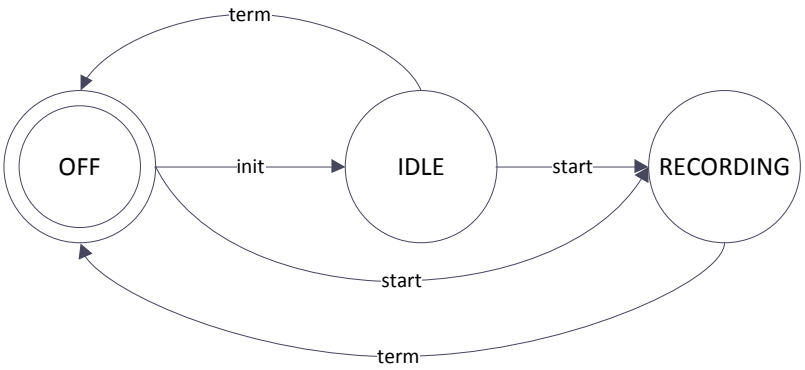


Figura 3.30 - Máquina de estados de una entrada de A/V

La figura anterior muestra la abstracción lógica donde minimicé el funcionamiento de una entrada de A/V para cualquier usuario o sistema externo que facilitan su uso y comprensión.

Cuando inicia la sesión de grabación, la entrada se encuentra en el estado de *apagado* (***OFF***), donde no se está ejecutando acción alguna. Si la máquina de estados es alimentada con el mensaje ***init*** (mensaje de inicialización), se crea y configura la entrada de acuerdo a los valores de configuración respectivos ubicados en el registro de configuración. En este mismo estado, una vez inicializada correctamente la entrada, se comienza a transmitir un streaming de video y audio para su visualización previa sin grabar a archivo. Cambiando el estado del autómata a *inerte* (***IDLE***). El streaming de audio y video lo proporciona el SDK de Optibase y es llevado a cabo bajo la transmisión de MPEG-4 ver 1. a través del protocolo RTP.

El mensaje ***start*** puede ser ingresado ya sea desde el estado OFF o IDLE, cambiando el estado del autómata a *grabando* (***RECORDING***), donde además de inicializar y configurar la entrada, transmitir la visualización del audio y video, se comienza a grabar la señal en un archivo multimedia con el nombre correspondiente de acuerdo a la convención establecida en el diseño.

Para finalizar la entrada, la máquina de estados se puede llevar al estado OFF directamente desde cualquiera de los dos estados IDLE y RECORDING.

Es importante contemplar la existencia de un estado de error en cualquier máquina de estados, por muy sencilla que ésta resulte, el estado de error no lo consideré en esta descripción por términos de simplicidad, sin embargo, no lo ignoré durante el desarrollo.

Dentro del servidor de captura, también consideré la existencia de un proceso que coteje el cambio de segmento o archivo durante la grabación. Un cambio de segmento lo obtuve parando la grabación del archivo y comenzando a grabar en un archivo nuevo. Es por ello, que esta acción coloquialmente hablando se llama *corte de archivo*. Dentro del sistema, por defecto, el cambio de segmento lo predeterminé para realizarse cada hora, sin embargo, el usuario lo puede configurar con un valor de 1 minuto hasta 4 horas como máximo.

Aprovechando la existencia del proceso que cuestiona el tiempo de grabación por entradas, me resultó conveniente agregar un procedimiento de rutina que verifique si la entrada asignada se encuentre trabajando correctamente, dentro del mismo proceso. De esta forma, sería posible detectar un error en caso de que éste ocurriera, y así reaccionar el programa podría corregirlo automáticamente.

Luego entonces, internamente la abstracción lógica de una entrada de captura la consideré como una máquina de estados, que es más compleja que la que se mostró al inicio de esta sección, como la que se muestra en la siguiente figura:

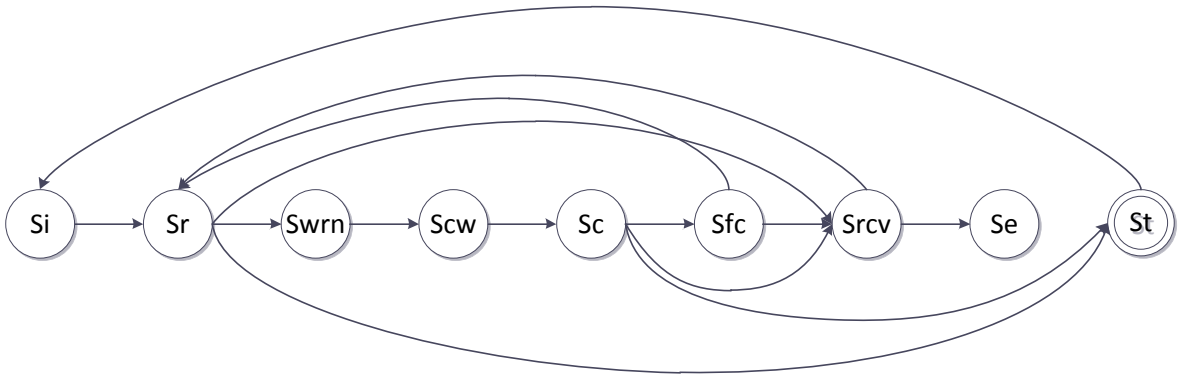


Figura 3.31 - Máquina de estados interna de una entrada

Donde:

So	*Apagado (**O**ff):*
Si	*Inerte (**I**dle)*
Sr	*Grabación (**R**ecording)*
Swrn	Alerta al corte (**W**a**RN**ing)
Scw	Espera al corte (**C**ut **W**ait)
Sc	Corte (**C**ut)
Sfc	Validación de archivo (**F**ile **C**heck)
Srcv	Recuperación (**R**e**C**o**V**ery)
Se	Error crítico (**E**rror)
St	Detenido (s**T**opped)

Los primeros tres estados del listado son los mismos que le muestro a los clientes externos, y el cambio de estados entre ellos ocurre de la misma manera en la que se describió al inicio de la sección.

El resto de los estados los utilicé para buscar la existencia de un error durante la grabación y cambiar de segmento automáticamente. El cambio de estado lo ejecuta la máquina de estados del proceso verificador, el cual se ejecuta en paralelo al proceso de grabación. Durante la ejecución de este proceso se inspeccionan tanto el tiempo de grabación del segmento en curso como el incremento del tamaño del archivo.

Como nota adicional al lector, dado que el proceso de verificación es totalmente interno, cualquier error que ocurra durante la grabación, junto con la posible causa que lo originó se reporta, en el archivo *FissionMonitorError.log,* ubiqué dentro del mismo directorio en el que se encuentra el ejecutable *FissionMonitorServer.exe*.

En el siguiente diagrama de flujo describo con mayor detalle la manera en que se ejecuta el proceso verificador:

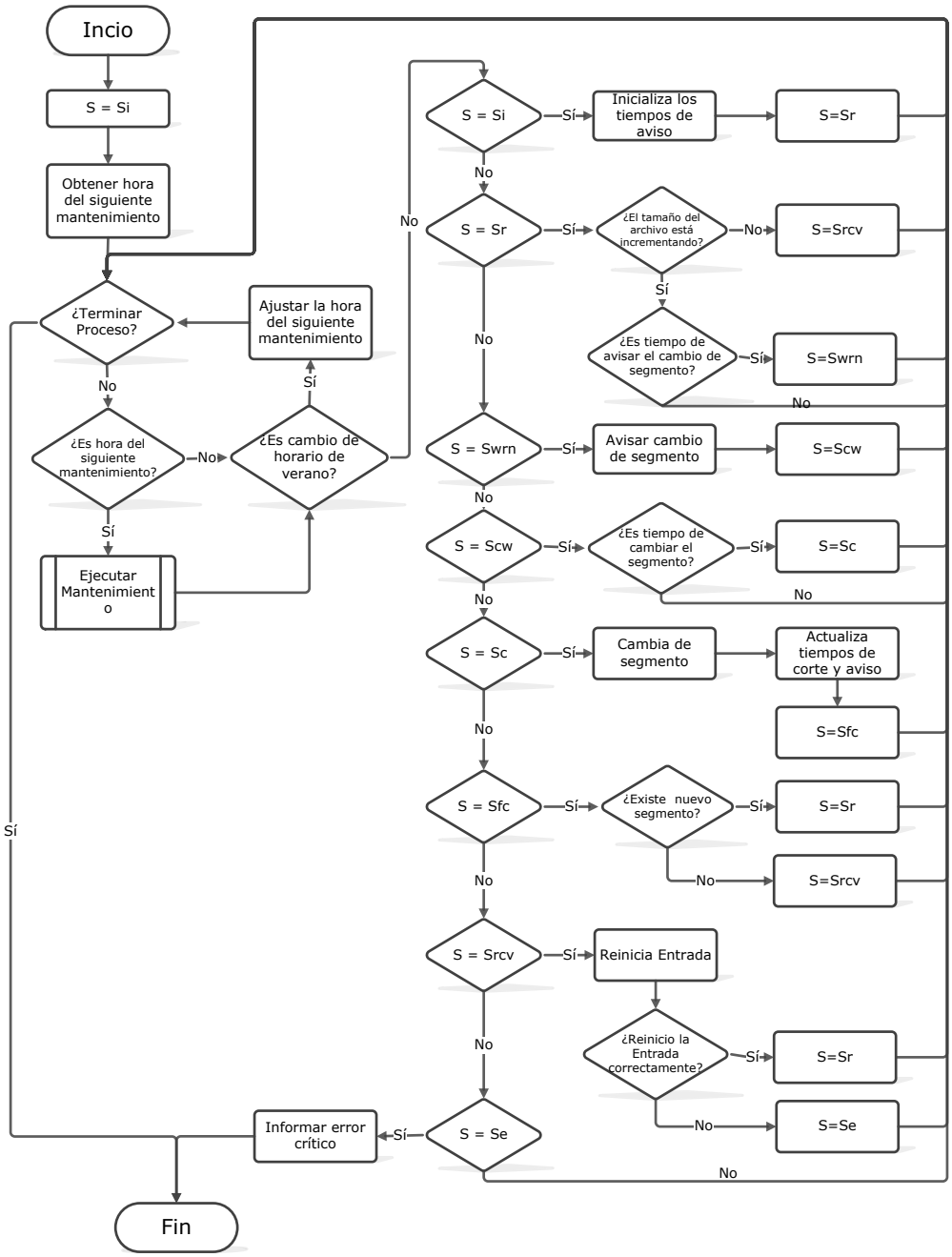


Figura 3.32 - Diagrama de flujo del proceso de verificación

Protocolo de comunicación

Para comunicarse con el módulo Server desde un medio externo diseñé un protocolo de comunicación sobre sockets UDP. El funcionamiento del protocolo lo implementé a manera de línea de comando, para facilitar su interacción con el usuario final. De esta forma pude eliminar la dependencia del módulo Fission Monitor Client para realizar pruebas durante la programación del mismo, Con esto también pude ofrecer al usuario la ventaja de poder seguir utilizando el servidor en caso de que el módulo cliente falle.

Debido a que en el protocolo UDP no cuenta con un control de recepción, y por lo tanto, de colisión de paquetes tan sofisticado como el TCP, para implementar la comunicación síncrona en este medio decidí realizar la comunicación de un canal con pares de puertos unilaterales, es decir, establecer un puerto para la recepción de comandos y otro para el envío de la respuesta, eliminando de estar forma cualquier posible colisión de los mensajes.

Tomando ventaja de que el producto se instalaría en un ambiente controlado, en otras palabras, el ordenador que hospeda al software solamente se utilizaría como un sistema de grabación del producto que desarrollé, me fue posible asignar los puertos de comunicación de una manera arbitraria como se muestra en la siguiente tabla.

Tabla 3.3 - Puertos UDP utilizados

Puerto UDP	E/S	Descripción
12030	Recepción	Inicialización y terminación de sistema. Puerto predeterminado para la manipulación por CLI
12031	Envío	
12040 + 2*n*	Recepción	Comandos de grabación para la entrada *n*
12041 + 2*n*	Envío	
12050 + 2*n*	Recepción	Configuración de mantenimiento y grabación para la entrada *n*
12051 + 2*n*	Envío	
12100 + n	Envío	Contadores Tiempo de Segmento de la entrada *n*

Primero cabe aclarar que en todos los puertos de entrada se aceptan todos los comandos del protocolo de comunicación que establecí para el producto. Tomando esa afirmación como premisa de la comunicación, el primer par de puertos de la tabla los asigné para inicializar y terminar una sesión. Estos puertos también se pueden utilizar para realizar pruebas al sistema o comenzar la comunicación desde un sistema externo.

El segundo par de puertos los determiné para las operaciones de control a la grabación. Establecí un par específico para cada entrada. Esta asignación está basada en la siguiente regla: *12040+2n* para los puertos para la recepción de comandos y *12041+2n* para el envío de las respuestas, donde *n* es el índice de la entrada a manipular e inicia con *n=0* para la primer entrada. Por ejemplo, para controlar la segunda entrada, que tiene como índice *n=1*, los puertos correspondientes serían 12042 para la recepción de comandos y 12043 para la respuesta correspondiente. Como nota mnemotécnica, tanto para el usuario como el desarrollador, contemplé los puertos pares para la recepción y el impar consecutivo para la transmisión de la respuesta.

Los puertos de configuración de la entrada, los fijé de la misma forma que los utilizados para la manipulación, pero iniciando a partir de los puertos 12050 y 12051 para la primer entrada.

Finalmente, los puertos asignados al envío de los contadores del tiempo transcurrido durante la grabación del segmento en curso por entrada se determinaron en base a la siguiente regla: *12100+n* donde *n* es el índice de la entrada. La unidad de los contadores es en segundos.

El diseño inicial que hice del protocolo solamente contempla la comunicación hacia un cliente por sesión, ya que dentro de los requerimientos solo se consideró que tanto el módulo *Client* como *Server* estarían ubicados dentro del mismo ordenador.

Uso Común de los Comandos

Los comandos recibidos por el servidor los agrupé en tres diferentes categorías: ejecución, configuración y consulta.

Los comandos de la categoría de ejecución son aquellos que afectan directamente la grabación, es decir, afectan el estado en el que se encuentra la entrada. Los comandos de configuración, realizan cambios en la configuración de la grabación y del mantenimiento de las unidades de almacenamiento. Recordando en todo momento que dentro del protocolo el índice de las entradas comienza a partir de *n=0*, hasta *n=k-1*, donde *n* es el índice de la entrada y *k* es el número de entradas existentes en el servidor.

A continuación quiero presentar los comandos más relevantes que permiten el inicio de un servicio de captura, su configuración para comenzar a grabar, hacer un corte de archivo y terminar el servicio. La documentación de todos los comandos implementados dentro del protocolo para el manejo del servicio se ubican en el Apéndice C.

Inicialización de una Entrada

Para inicializar una sesión de captura primero se debe de enviar el comando de ***init*** seguido del índice de la entrada con la que se desea comenzar a trabajar.

```
>init 0
ok
```

En caso de no poder ejecutar el comando correctamente se recibirá un mensaje de error. En este punto ya se tiene habilitado el streaming con el audio y video de la entrada para su visualización previa.

Manipulación de Captura para una Entrada

Una vez que se tiene completamente inicializada la entrada en la cual se va a capturar, se envía el comando `start` para comenzar a grabar. Si la grabación comienza exitosamente, el servidor responde con un eco del comando utilizado, seguido de un espacio y el nombre con ruta del archivo en el cual se están grabando las señales:

```
>start 0
start C:\Capturas\In1_Canal1_20120128_033000.mp4
```

En caso de que exista un error al momento de comenzar a grabar el audio y video, el sistema responde con un mensaje de error que consiste de la palabra `error` seguido de un espacio y el código de error correspondiente.

Una vez que la entrada se encuentra capturando, se pueden realizar dos acciones, cambiar el segmento de grabación o terminar la grabación. Para realizar un *corte de archivo* se envía el comando `cut` junto con el índice de la entrada:

```
>cut 0
cut C:\Capturas\In1_Canal1_20120128_054523.mp4
```

De la misma forma que el comando `start`, si la operación es exitosa regresa un eco del comando utilizado, un espacio y el nombre completo del nuevo archivo en el cual se está grabando. En caso contrario se recibe un mensaje de error

Finalmente, para detener una grabación se utiliza el comando *stop* seguido del índice de la entrada en la cual se desea realizar la acción:

```
>stop 0
stop C:\Capturas\In1_Canal1_20120128_230156.mp4
```

Al igual que los últimos dos comandos explicados, en caso positivo responde con un eco del comando *stop* seguido del nombre del archivo asignado a la grabación de la entrada en cuestión.

Cierre de una Entrada

Al igual que el resto de los comandos explicados previamente para cerrar la sesión de grabación de una entrada en específico, se usa el comando *term* de la siguiente manera:

```
>term 0
ok
```

Parser de entrada

Los comandos recibidos por el servidor, como los que se acaban de mencionar, son procesados por un parser programado específicamente para el protocolo de comunicación que utiliza el módulo servidor. Un parser se puede definir como:

Componente de un programa que analiza la sintaxis de un texto y extrae la información basada en una gramática formal.

En este caso en particular, el texto es la cadena de caracteres recibida a través del comando y la información es la acción que el usuario desea que ejecute el servidor. Al momento de recibir el comando, el parser valida la sintaxis y si es correcta, se procesa para obtener los parámetros necesarios para ejecutar la acción y posteriormente ser enviados al intérprete**.** Éste último define que proceso ejecutor es el adecuado para efectuar el comando recibido.

Si un error es detectado en la sintaxis del comando, el intérprete responde con un mensaje de error al usuario o sistema externo. La localización de este tipo de errores a nivel de sintaxis contemplé que es muy útil para el cliente, ya que de esta forma puede corregir con mayor facilidad su fallo.

El propósito del parser que implementé es el determinar si un mensaje recibido por medio de sockets corresponde a un comando con el formato correcto de acuerdo a la definición del protocolo.

Para facilitar el procesamiento de los comandos, los catalogué en cuatro tipos, de acuerdo a sus parámetros:

Tabla 3.4 - Tipos de comando por parámetro

Tipo	Tokens	Número de Parámetros	Descripción Parámetros	Errores Posibles
1	1	0	N/A	E0
2	2	1	Numérico	E0, E1
3	2	1	Alfanumérico	E0, E2
4	4	3	1. Numérico 2. Alfanumérico comenzando con un guión (-) 3. Numérico o alfanumérico y el rango depende del 2º parámetro	E0, E5, E3, E4

Al igual que las entradas de captura, el parser lo diseñé en base a una máquina de estados que recibe como entrada cada uno de los tokens del comando recibido. Esto a su vez facilitó su programación ya que la implementación de una máquina de estados se puede realizar fácilmente definiendo la entrada como una estructura de datos, el estado como otra estructura de datos que alimentan un objeto y toma decisiones por medio de un *switch-case* en el caso de la programación en VC++.

En el siguiente diagrama de flujo junto con la tabla #, muestro como diseñé el parser junto con los posibles errores que me puede enviar si el comando no fue propiamente construido.

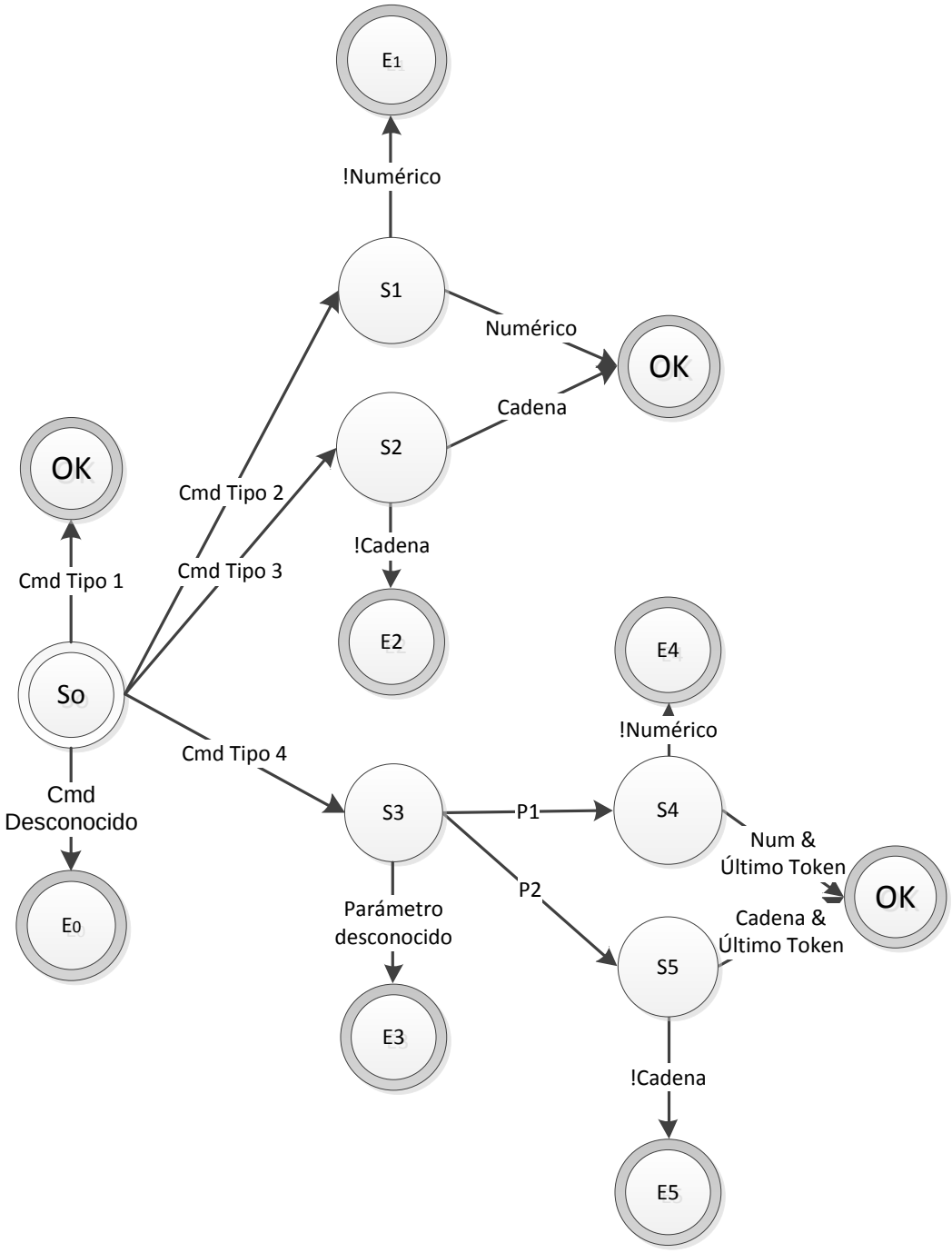


Figura 3.33 - Parser de recepción de mensajes

Donde los posibles errores del autómata anterior se describen en la tabla:

Tabla 3.5 - Errores que detecta el parser

Error	Descripción
E0	El primer token es un comando desconocido
E1	El comando tipo 2 y el 2º token no es numérico
E2	El comando es tipo 3 y el 3er token no es cadena de caracteres
E3	El tipo de parámetro es desconocido
E4	El valor del parámetro requiere que sea numérico y no lo es
E5	El valor del parámetro requiere que sea una cadena de caracteres y no lo es

TDA de captura

Tomando en consideración las acciones que la entrada debe de ejecutar para cumplir con el autómata descrito en la sección 6.2.1, la entrada se puede abstraer en el siguiente Tipo de Dato Abstracto (TDA):

PlantillaDeCaptura
▪ Inicializar(*Parámetros de Inicialización*) ▪ Terminar() ▪ AbrirSesión(*Parámetros Incio de Sesión*) ▪ ObtenerTiempoGrabación() ▪ MuestraEntrada(*Apuntador donde se muestra la entrada*) ▪ DeténMostrarEntrada(*Apuntador donde se muestra la entrada*) ▪ ComenzarGrabación() ▪ CambiaSegmento() ▪ DeténGrabación() ▪ ReiniciaSesión() ▪ CerrarSesión()

Con el TDA de captura ya definido, la implementación en el lenguaje VC++ se vuelve una tarea sencilla, ya que al aprovechar las ventajas que ofrece el lenguaje, la implementación del TDA se remite a la programación de una clase abstracta a partir de la cuál se deriva una clase que utiliza el SDK de Optibase para controlar el hardware.

La ventaja que conseguí, al implementar el TDA en una clase abstracta, es que no dependía de un SDK o API, con lo que obtuve una clase *comodín* que lista los huecos de las tareas por cubrir para poder realizar una captura, ofreciendo mediante ésta una guía en la programación para una nueva clase que implemente el uso de otra tarjeta de captura en caso de que algún cliente lo llegara a solicitar o la empresa así lo requiera.

La definición de la clase abstracta de captura quedaría de la siguiente forma en el lenguaje Visual C++:

```
class CfRecorderTemplate
{
public:
    CfRecorderTemplate(void);
    virtual ~CfRecorderTemplate(void);

    virtual bool          Initialize(FInitStruct initStruct) = 0;
    virtual void          Terminate(void) = 0;
```

```
    virtual bool          StartSession(FOpSessionStruct openStruct) = 0;
    virtual bool          RestartSession(void) = 0;
    virtual void          StopSession(void) = 0;

    virtual UINT64 GetSegmentRecTime(void) = 0;

    virtual bool          DisplayInputOverlay(FInputOverlay* pOverlay) = 0;
    virtual bool          HideInputOverlay(FInputOverlay* pOverlay) = 0;
    virtual bool          StartRecording(CfString newFile) = 0;
    virtual bool          StopRecording(CfString lastFile) = 0;
    virtual bool          CutAndContinue(CfString newFile) = 0;
    virtual bool          GetCurrentFile(CfString curFile) = 0;
};
```

Donde cada método virtual es un comodín para ser adecuado al funcionamiento específico del hardware a utilizar para la solución.

Programación paralela

Para tratar el tema de la programación de sistemas paralelos es importante explicar las dos arquitecturas paralelas existentes con mayor importancia en la industria en la actualidad: la arquitectura de Memoria Distribuida (o Multicomputador) y la arquitectura de Memoria Compartida (o Multiprocesador).

Arquitectura Multicomputador

Cada procesador u ordenador tiene su propia memoria para el procesamiento. Los datos son compartidos por mensaje ya que estos se encuentran ubicados en la memoria del procesador u ordenador. Su programación requiere impulsar la localidad de los datos para reducir la cantidad de mensajes y así hacer más eficiente el procesamiento, lo cual puede consumir gran tiempo del desarrollo. Sin embargo, este tipo de arquitectura cuenta con una mejor escalabilidad que la arquitectura de memoria compartida, ya que su instalación es transparente y dependiendo del sistema puede ser ininterrumpido.

Por otro lado, una gran desventaja que este modelo presenta es la transferencia que los datos tienen, ya que éstos tienen que ser transferidos físicamente a la memoria local de cada nodo antes de su ejecución, lo cual constituye un trabajo adicional significativo y al finalizar el procesamiento los resultados tienen que ser trasmitidos de cada nodo al sistema huésped. En otras palabras, la distribución de datos resulta difícil de implementar

Arquitectura Multiprocesador

En un sistema multiprocesador, cada procesador tiene acceso a toda la memoria, en otras palabras, existe un espacio de direccionamiento compartido. Todos los procesadores se encuentran comunicados con la memoria principal de la misma forma y la accedan por el mismo ducto (pipe). Este tipo de arquitectura es predominante en el mercado de los ordenadores caseros, así como en servidores con fines empresariales basados en la arquitectura x8086. Es precisamente este el tipo de arquitectura con la que cuentan los servidores a utilizar como huésped del sistema a implementar.

Durante la programación es importante evitar que los procesadores accedan de manera simultánea a las regiones de memoria compartida, ya que esto ocasionaría un error dentro del sistema, la cual sería fatal para la aplicación. Si dos o más procesadores

desean acceder una variable de memoria compartida deben de establecer procedimientos de exclusión mutua para acceder los datos compartidos. Para llevar a cabo esta tarea, existen diversos mecanismos como el uso de candados o semáforos y mecanismos de sincronización explícitos o implícitos. En este modelo de programación paralela la comunicación es implícita, debido a que se usa un solo espacio de direccionamiento.

Para comenzar la explicación de la programación multi-hilos, paradigma utilizado con gran popularidad sobre esta arquitectura, es necesario aclarar lo que es un hilo. Un hilo (*thread*) es un flujo simple de control secuencial. Se define como un proceso que comparte el mismo espacio de memoria y variables globales con el proceso que lo creó.

En un lenguaje de alto nivel, como lo es Visual C++, se programa un hilo al igual que se programa un procedimiento que utiliza la pila de la forma tradicional. En el caso particular donde solo existe un hilo dentro del programa, en cualquier instante existe solamente un punto de ejecución. Por lo tanto la programación es transparente y el programador no requiere aprender nada nuevo más que el uso de las bibliotecas para la creación y manipulación de hilos. Esto demuestra una de las ventajas que ofrece MFC, ya que soporta nativamente la programación multi-hilos y por lo tanto la implementación de hilos se vuelve una tarea simple.

Por otro lado, cuando se tienen múltiples hilos dentro de un programa, se tiene que en cualquier instante se existen múltiples puntos de ejecución, donde cada punto corresponde a cada hilo. El programador puede considerar que cada hilo como se está ejecutando simultáneamente, por lo que se vuelve tarea del programador decidir cuando y y en que punto del programa se deben crear los hilos. No obstante, debe de tomar en cuenta que en la realidad el ordenador no ejecuta todos los hilos simultáneamente, ya que esto depende de la capacidad de procesamiento de cada núcleo de los procesadores presentes en el sistema. Es por ello que el acceso a datos compartidos tiene que realizarse de acuerdo a las siguientes reglas:

1. La lectura múltiple del valor de una variable no causa conflicto
2. La escritura a una variable compartida tiene que hacerse por medio de exclusión mutua
3. El acceso a secciones críticas se controla mediante una variable compartida

Administrador de hardware

El paradigma de programación multi-hilos es el método que mejor se adapta a las necesidades del software a desarrollar, ya que el SDK de Optibase no está implementado para funcionar de una manera segura en un paradigma de programación paralela. Por este motivo, el desarrollo de una clase que sincronice los accesos a las instancias de los objetos que controlan las entradas de los sistemas, evita la corrupción de la memoria compartida de la tarjeta capturadora. Dicha clase estará encargada de sincronizar las instancias de la clase CRte, clase derivada de la clase abstracta *CfRecorderTemplate,* que representan cada una de las entradas de señal A/V dentro del sistema.

La clase de sincronización dentro del servidor se nombró *CfOptibaseManager* y su implementación consiste en encapsular todos los métodos descritos en el TDA de acuerdo al siguiente pseudocódigo:

```
tipo_regreso CfOptibaseManager::MetodoAEncapsular(int entrada, parámetros…)
{
	Tipo_regreso resultado;
	CSingleLock candado(m_SeccionCritica);

	candado.Lock();
```

```
        resultado = arregloEntradas[entrada]->Metodo(parámetros…);
        candado.Unlock();

        return resultado;
}
```

Ampliando la descripción del código anterior, para evitar que más de un proceso a la vez acceda la memoria compartida de las bibliotecas dinámicas que proporciona Optibase en su SDK utilicé candados en cada una de las secciones críticas de las clases *CRte*.

Interfaz gráfica de usuario

Dado que este módulo funciona como un servicio, la interfaz gráfica se llevo al mínimo, ya que cualquier interacción significativa se realiza a través módulo cliente.

La interfaz consiste solamente de un ícono en la barra de tareas de Windows, un menú contextual y un diálogo que muestra el estado de cada entrada. El ícono tiene 2 funciones, la de mostrar el estado del servidor de una manera iconográfica y la de permitir una interacción con el servidor. Por medio del ícono que se muestre, se puede saber si el servidor se encuentra en cualquiera de los siguientes estados:

Ícono	Estado
	Inicializando
	En funcionamiento
	Error

Figura 3.34 - Íconos de estado de Fission Monitor Server

Existen dos formas de interactuar con el ícono, la primera es mediante un doble clic izquierdo el cual abrirá el diálogo de estado de las entradas del servidor y la segunda es mediante un clic derecho el cuál mostrará un menú contextual que permite realizar un mínimo de acciones sobre el servidor.

El menú contextual está formado por los siguientes elementos:

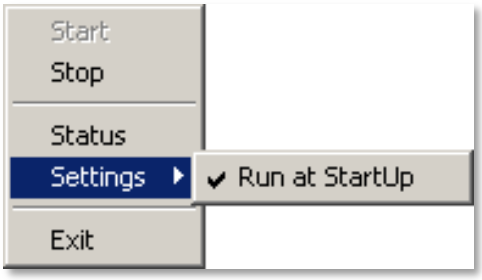


Figura 3.35 - Menú contextual del módulo Server

Start: Comienza a escuchar en todos los puertos de recepción de comandos

Stop: Deja de escuchar en los puertos de recepción de comandos y detiene las grabaciones que se estén llevando a cabo

Status: Muestra el diálogo de estado de las entradas del sistema

Run at StartUp: Establece si se desea que el servicio de captura se ejecute cuando inicia el sistema operativo

Exit: Cierra la aplicación

Para ejemplificar el diálogo de estado que presenta la fase en el que se encuentra cada entrada de captura del sistema se presenta a continuación una captura de pantalla del diálogo correspondiente:

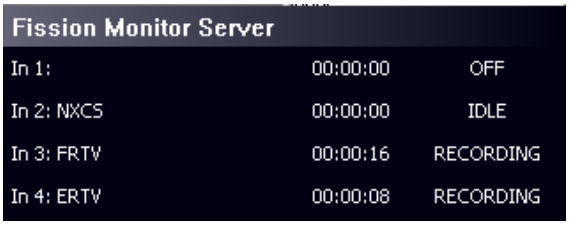


Figura 3.36 - Diálogo de estado del módulo Server

El tamaño del diálogo se ajusta de acuerdo al número de entradas que posee el sistema. Los posibles estados a mostrar por entrada son:

Init: Se ha inicializado la entrada física en el hardware y el servidor está listo para recibir los comandos correspondientes

Idle: Se trasmiten las señales de A/V de la entrada sin grabar.

Recording: Se transmiten y graban a archivo las señales de A/V de la entrada.

Error: Existe un error crítico durante la ejecución de alguna acción en la entrada y ha cesado de funcionar correctamente.

Solamente cuando la entrada se encuentra en el estado de *Recording*, aparecen los contadores del tiempo transcurrido de la grabación del segmento actual.

3.5.4. Fission Monitor Upkeeper

Una vez implementado el servicio de captura como se describió en la sección anterior. Para poder lograr que la grabación califique como ininterrumpida, es necesario crear un proceso que tenga como tarea principal mantener un espacio libre mínimo en las unidades de almacenamiento utilizadas por el sistema para capturar. De esta forma, se asegura que la grabación no se interrumpa por falta de espacio libre en disco.

Para llevar a cabo esta tarea, la manipulación de archivos es indispensable. Por ello resulta conveniente agregar el proceso de copiado de archivos de la ruta donde se realiza la captura al almacenamiento masivo a largo plazo, en caso de ser solicitado por el usuario. Cumpliendo lo estipulado en los requisitos enlistados en el capítulo de Requerimientos y contemplado en el capítulo de Diseño. Por lo tanto, las tareas a ejecutar en este módulo son:

1. Copiar archivos capturados al almacenamiento a largo plazo
2. Eliminar los archivos expirados
3. Liberar espacio libre en las unidades de almacenamiento

El primer punto se refiere a copiar los archivos recién capturados y que no se encuentran en el almacenamiento a largo plazo. Esta tarea es opcional y el usuario final es quien determina si se realiza su ejecución.

De la misma manera, el usuario puede establecer el máximo de días que un archivo tiene de vida dentro del sistema, si ese tiempo de vida es excedido, dicho archivo se considera expirado y será eliminado tanto del almacenamiento a largo plazo como de la ruta utilizada para capturar. A esto se refiere la segunda tarea de la lista.

Por último, la tercera tarea es la más importante dentro del módulo de mantenimiento, ya que al ejecutarla correctamente, el módulo *Server* es capaz de grabar en todo momento, brindándole el espacio suficiente en disco para almacenar los archivos capturados. El orden de eliminación se dará por la antigüedad del archivo.

El siguiente diagrama de flujo se describe de manera general la ejecución del *Upkeeper*:

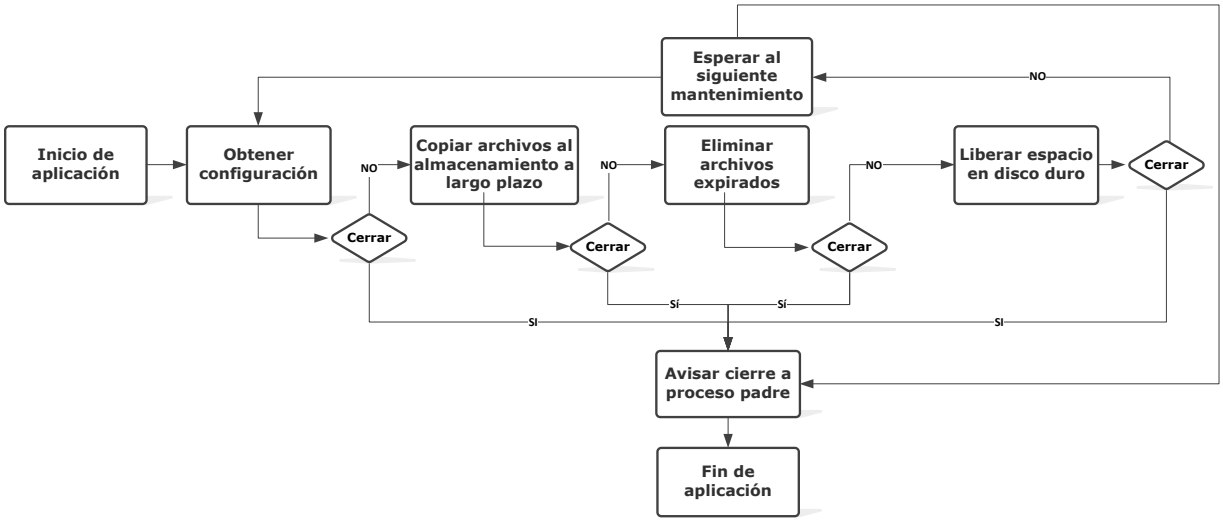


Figura 3.37 - Diagrama de flujo operación Upkeeper

La aplicación Fission Monitor Upkeeper es ejecutada por el programa Fission Monitor Server, el cual a su vez envía un mensaje de cerrar aplicación por medio de mensajes de Windows de memoria compartida de *CopyData* como se mencionó el capítulo de Diseño.

Para ejecutar el proceso de borrado de archivos se sigue el algoritmo descrito por el siguiente pseudocódigo:

```
1. Obtener listado ordenado alfabéticamente de los archivos de cada uno de los directorios
   a ser procesados
2. Obtener tiempo de vida de cada directorio obtenido en 1.
3. Por cada listado obtenido en 1.
4. Mientras la fecha y hora del primer elemento de la lista exceda el tiempo de vida
   asignado al directorio
   4.1. Borrar el archivo descrito en el primer elemento de la lista
   4.2. Eliminar primer elemento de la lista
5. Fin
```

En el primer punto los directorios mencionados son los de captura y almacenamiento a largo plazo de cada entrada. Los directorios de captura siempre van a estar

De la misma manera para procesar la liberación de espacio en disco indicado en el punto 3 se sigue el algoritmo descrito por el siguiente pseudocódigo:

```
1. Obtener listado ordenado de los archivos ubicados en cada uno de los directorios a ser
   procesados
2. Obtener un listado de los directorios raíz utilizados por las listas de 1.
3. Mezclar las listas de los directorios que tengan el directorio raíz en común por medio
   del método de Ordenamiento por Mezcla.
4. Obtener espacio libre mínimo (ELMR) requerido por el usuario de cada lista obtenida en 3.
5. Por cada lista obtenida en 3.
   5.1. Mientras el espacio libre del directorio raíz de la lista en 4 es menor a su ELMR
       5.1.1. Borrar el archivo descrito en el primer elemento de la lista
       5.1.2. Eliminar primer elemento de la lista
6. Fin
```

3.5.5 Fission Monitor Client

El módulo Client, conforme al diseño inicial, es la aplicación que funciona como interfaz de usuario de todo el sistema. Desde éste se pueden manipular las grabaciones, configurar la manera de grabar, configurar el mantenimiento de las unidades de disco, visualizar el contenido del registro de la grabación y exportar videoclips del mismo registro.

Flujo de la interfaz

Para obtener una idea más clara del funcionamiento se presenta un diagrama que muestra el orden en que son llamados los diálogos para interactuar con el sistema:

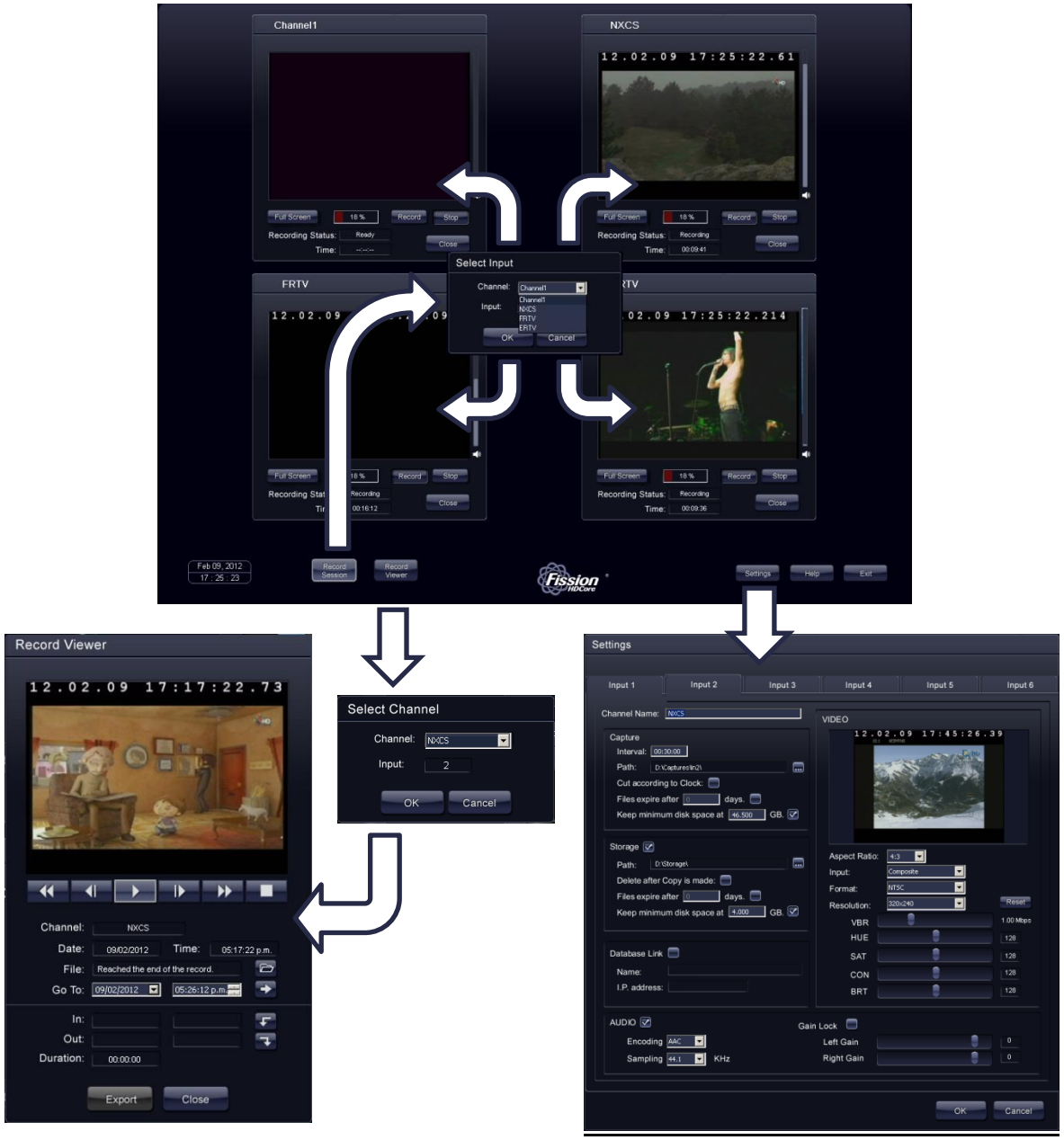


Figura 3.38 - - Flujo de diálogos del módulo Client

Sesión de grabación

Al diálogo de la sesión de captura no le adapté mayores cambios durante la implementación, salvo la adición de los siguientes controles:

- Porcentaje de espacio libre en el disco en el que está capturando la entrada
- Etiqueta del estado de la entrada
- Etiqueta del tiempo de grabación del segmento en curso
- Control de volumen del audio de la entrada.

Figura 3.39 - Sesión de grabación final

La información del control que muestra el espacio libre en disco la obtuve por medio de llamadas al sistema operativo. El estado de la grabación así como las acciones de los botones *Record* y *Stop* se llevan a cabo mediante el envío de los comandos ***curstat***, ***start*** y ***stop*** respectivamente a través del socket UDP con su puerto correspondiente de acuerdo a la Tabla 3.3 - Puertos UDP que definí previamente en la sección **6.2.2.**.

El tiempo de captura del segmento en grabación etiquetado bajo *Time* se obtiene a partir de los mensajes recibidos por el puerto UDP correspondiente como también se describió en la Tabla 3.3 - Puertos UDP (pág. 61).

El video que se muestra por entrada es el resultado de una gráfica de reproducción en DirectShow basado en el SDP adquirido del servidor a través del comando ***sdp***. Cuando el módulo cliente ha obtenido la cadena de caracteres que describe la sesión de streaming de video, ésta es almacenada en un archivo bajo en nombre **Input**<*num_entrada*>**.sdp** guardado en el mismo directorio que el ejecutable *Fission Monitor Cliente.exe*.

A partir de este archivo con extensión **.sdp** se crea la gráfica de reproducción del streaming de video proveniente del módulo *Server*. Es importante recalcar que el ordenador debe de contar con la instalación de la aplicación ***Optibase Player 400*** y su correcta configuración como se indica en el Apéndice D. Esta aplicación cuenta con los filtros de DirectShow necesarios para crear la gráfica a partir de un archivo *SDP* utilizando la *Conexión Inteligente* llamando el método *RenderFile* de la instancia de *IGraphBuilder* que representa el *Filter Graph Manager*.

Visualizador de video

El visualizador de los archivos capturados sufrió muy pocos cambios con respecto al diseño original como se puede observar en la *Figura 3.40*. Solamente se agregaron:

- Texto que indica de que canal se está viendo el contenido
- Texto de fecha y hora que se están visualizando del contenido
- Texto del archivo multimedia que está siendo visto
- Botón que abre el muestra *Explorador de Windows* con la ubicación en la que se encuentra el archivo seleccionado.

Figura 3.40 - Visualizador de captura final

Para visualizar con mayor claridad la manera en la que el software toma decisiones sobre el archivo a mostrar o la forma en que se exporta, resulta conveniente utilizar un ejemplo. Se pueden considerar a los segmentos del registro de captura como bloques sobre una línea del tiempo, en este ejemplo se considera que los segmentos tienen una duración de 30 minutos.

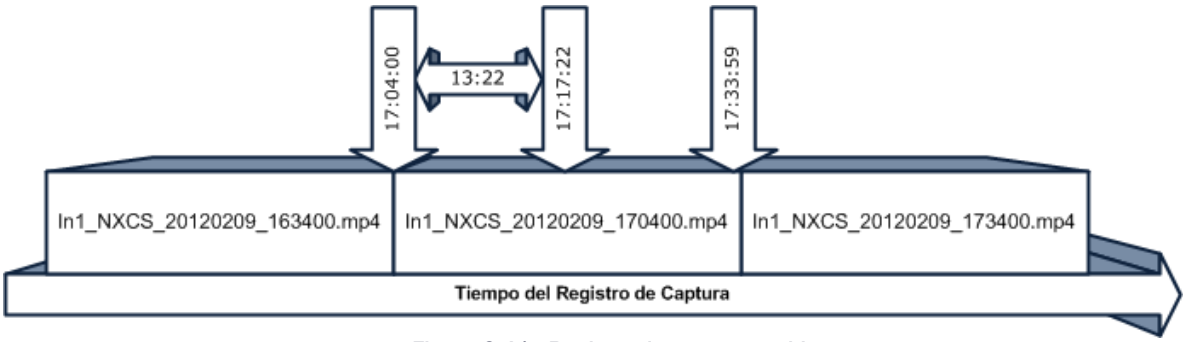


Figura 3.41 - Registro de captura en bloques

Ahora bien, el usuario desea visualizar las 17:17:24 horas del día 9 de Febrero del 2012 dentro de este registro. Para llevar a cabo dicha tarea, el visualizador tras haber realizado la búsqueda (SEEK) del archivo determina que el archivo que lo contiene es el In1_NXCD_20120209_170400.mp4. Basado en el criterio de nomenclatura del archivo, este archivo el audio y video del canal NXCS a partir de las 5:04:00 P.M. del día descrito en la búsqueda y abarca hasta las 5:33:59 P.M. del mismo día.

Una vez obtenido el archivo de video y su tiempo de inicio de grabación, se calcula el tiempo dentro del archivo que corresponde al tiempo de búsqueda. Este se obtiene de la siguiente forma:

[Tiempo de Visualización] = [Tiempo de Búsqueda] – [Tiempo de Inicio de Archivo]

Para este ejemplo el tiempo de visualización es de 13:22 minutos. Al llegar a este punto el visualizador es capaz de crear una gráfica de reproducción para el archivo In1_NXCD_20120209_170400.mp4, haciendo uso de DirectShow como se describe en el Apéndice B, y adelantar la reproducción del video al minuto 13:22 para finalmente mostrarle al usuario el resultado.

A partir de este momento el usuario puede reproducir el video, pausarlo, adelantarlo rápidamente o retroceder dentro de los archivos capturados. Durante este momento de navegación en el registro de captura, el usuario ya puede comenzar el proceso de exportación.

Exportador de clips de video

Para exportar un video del registro, es imperativo que el usuario marque mediante el diálogo de visualización la fecha y hora del registro en la que el usuario desea que empiece la exportación, mediante el botón IN, y también es necesaria la fecha y hora en la que desea que termine la exportación, mediante el botón OUT.

Cuando el programa obtiene esos parámetros, verifica que el videoclip pueda ser creado correctamente, solicita al usuario el nombre y ubicación del archivo que contiene la exportación con el diálogo de la *Figura 3.42*, crea una lista de exportación en XML y ejecuta el módulo *Fission Monitor Exporter* para crear el videoclip basado en dicha lista.

Figura 3.42 - Diálogo final de exportación

Retomando la visión de los archivos capturados en forma de bloques sobre una línea del tiempo, los posibles casos con los que se puede enfrentar el módulo al crear la lista de exportación son cinco:

Caso 1

En el primer caso los tiempos de entrada y salida de la exportación se encuentran dentro del mismo archivo y por lo tanto la exportación se realiza en base a un segmento del archivo que contiene el tiempo especificado.

Figura 3.43 - Exportación segmento en un archivo

Caso 2

En el segundo caso, la entrada se encuentra en un archivo y la salida en otro, con la posibilidad de que existan entre ellos. En este caso, se agregan n elementos, donde n es el número de archivos que contienen el video a exportar. El primer caso es una particularidad de este caso cuando n = 1. Sin embargo se considera necesario ser explícito para la comprensión del sistema.

Figura 3.44 - Exportación segmento en múltiples archivos

Caso 3

Este caso ocurre cuando no hay captura para la entrada de la exportación. Lo que el sistema hace es sugerir un remplazo de ese valor al usuario con el tiempo posterior más próximo existente dentro del registro. El rango de archivos es *n* >= 1.

Figura 3.45 - Exportación con entrada sin registro

Caso 4

Es un caso similar al tercer caso pero tratándose del tiempo de salida de la exportación. La sugerencia en esta ocasión es el tiempo anterior más próximo existente dentro del registro. De la misma forma el rango de archivos es *n* >= 1.

Figura 3.46 - Exportación con salida sin registro

Caso 5

El quinto caso implica la ocasión en que se tiene un hueco dentro del registro mayor a la tolerancia que esté configurada dentro del sistema. Cuando esto ocurre, el

programa le pregunta al usuario si desea continuar a pesar de la carencia de material capturado, en caso de que la respuesta del usuario sea afirmativa, se agrega un hueco dentro de la lista de exportación donde se mostrarán cuadros en negro sin audio durante ese lapso.

Figura 3.47 - Exportación con un hueco

Este mismo caso se puede presentar múltiples veces dentro de una misma exportación como se muestra en la figura:

Figura 3.48 - Exportación con múltiples huecos

Concluyendo, el caso 1 y 2 son totalmente excluyentes de cualquier otro caso durante una exportación. Sin embargo, el caso 3, 4 y 5 pueden ocurrir dentro de un mismo proceso de exportación. Estas eventualidades las consideré durante la implementación de la creación de la lista de exportación y su interacción con el usuario.

Lista de exportación

Cuando el usuario ya estableció los tiempos de entrada y salida de exportación del registro de captura se crea una *lista de exportación* a partir de la lista de archivos ordenados cronológicamente obtenidos por el método mencionado durante el análisis previo.

Para crear una lista de exportación decidí considerar a los segmentos del registro de captura como bloques que pueden ser incluidos total o parcialmente dentro de la exportación. Definiendo la lista de exportación con los siguientes elementos:

1. El tiempo de salida tiene que ser posterior al tiempo de entrada
2. La lista debe de tener al menos un elemento
3. Los elementos que la componen están definidos por los siguientes valores:
    a. Nombre del archivo
    b. Tiempo de inicio de exportación
    c. Tiempo de fin de exportación

Tomando ventaja de que la exportación la realiza el módulo Fission Monitor Exporter, la manera en que el módulo cliente le comunica que lista exportar es mediante un XML, ya sea por medio de una cadena o contenida en un archivo de texto.

El módulo *Client* le transmite al módulo *Exporter* la lista exportación a través de los argumento de ejecución del último programa. Si al momento de ejecutar el programa se incluye el parámetro ***–s***, esto indica que el argumento siguiente corresponde a la lista de exportación formateada en una cadena de caracteres formateada en XML definida por el siguiente DTD:

```
<!--ExportList>
<!ELEMENT FOAMExport (Save, FileList)>
<!ELEMENT Save (Path, Name, FullName)>
<!ELEMENT FileList (File*)>
<!ATTLIST Path #CDATA REQUIRED
Name           #CDATA  REQUIRED
FullName       #CDATA  REQUIRED
File           #CDATA  REQUIRED
InPos          #CDATA  REQUIRED
OutPos  #CDATA  REQUIRED
ClipLen #CDATA  REQUIRED >
```

Un ejemplo de una lista de exportación es la siguiente:

```
<FOAMExport>
        <Save>
                <Path>D:\Exports\</Path>
                <Name>ejemplo.asf</Name>
                <FullName>D:\Exports\ejemplo.asf</FullName>
        </Save>
        <FileList>
                <File>
                        <Name>D:\Captures\In2\In2_NXCS_20120220_191922.mp4</Name>
                        <InPos>770000000</InPos>
                        <OutPos>3002610000</OutPos>
                        <ClipLen>2232610000</ClipLen>}
                </File>
                <File>
                        <Name>D:\Captures\In2\In2_NXCS_20120220_192422.mp4</Name>
                        <InPos>0</InPos>
                        <OutPos>3016380000</OutPos>
                        <ClipLen>3016380000</ClipLen>
                </File>
        </FileList>
</FOAMExport>
```

Otra forma de alimentar al programa encargado de la exportación es mediante el parámetro **–f** que indica que el siguiente argumento corresponde al archivo XML que contiene la lista de exportación.

Exportación por medio de Microsoft Media Export Services

Una vez que el *Exporter* ha sido alimentado con la lista de exportación, este hace uso de los servicios de exportación de multimedia que ofrece DirectShow llamados Microsoft Exporting Services.

Describiendo su uso de una forma muy general, estos servicios trabajan con gráficas que en lugar de reproducir una salida a pantalla, como las utilizadas en el visualizador, escriben a un archivo codificando el video y audio de acuerdo a su configuración. Dado que es una herramienta desarrollada sobre DirectShow también se puede aprovechar la *Conexión Inteligente* de una gráfica.

La forma general en la que utilicé los servicios de exportación que ofrece Microsoft para dar una mejor idea de cómo el módulo crea los videoclips que le solicita el usuario es la siguiente:

Para comenzar a crear un videoclip es necesario crear una línea del tiempo mediante una instancia del objeto `IAMTimeline`. Esta línea de tiempo es inicializada mediante la adición de las pistas de video y audio a incluir en el videoclip. Las pistas son del tipo `IAMTimelineTrack`.

Las pistas son creadas en base a una secuencia de elementos que contienen el nombre del archivo, tiempo de entrada y tiempo de salida del segmento del archivo fuente. A su vez, estos elementos también deben de contener tiempos correspondientes en el video resultante de la exportación. Estos elementos son instancias del tipo `IAMTimelineObj` y su información se obtiene en base a la lista de exportación proveniente del módulo *Client* que describí anteriormente.

En el momento en que se cuenta con la línea del tiempo bien establecida con sus pistas propiamente creadas de acuerdo a lo que el usuario solicitó, es momento de crear la gráfica. En esta ocasión, la gráfica se crea por medio del objeto COM *IRenderEngine*. Este objeto representa a la gráfica de exportación así como el *IGraphBuilder* representa a la gráfica de reproducción de video. Este objeto es inicializado con la línea del tiempo previamente creada en base a la lista de exportación. Después se le agrega el filtro encargado de codificar el videoclip solicitado, que en el caso particular del sistema se trata de un ASFWriter.

Posteriormente, para crear la gráfica usando *Intelligent Connect*, en lugar de llamar el método *RenderFile* se llama al método *RenderStream* con los parámetros adecuados para cada pista incluida previamente en la instancia de `IAMTimeline`.

Finalmente cuando la gráfica se encuentra debidamente construida, se llama al método *Run* con lo que comenzará el proceso de exportación. Durante el proceso de exportación aparece un diálogo como el que se muestra en la Figura 49, que indica mediante una barra de progreso el avance del proceso y el nombre del archivo final. Este proceso puede ser cancelado en cualquier momento con lo que se borrará el avance parcial de la exportación.

Figura 3.49 - Diálogo de proceso de exportación

Configuración del sistema

La configuración del sistema se puede dividir en cinco categorías principales:

1. Captura
2. Almacenamiento a largo plazo
3. Base de Datos
4. Video
5. Audio.

Este diálogo sufrió muchos cambios con respecto a su diseño original. Mientras me pedían que el software cumpliera con una mayor funcionalidad, en la mayoría de los casos esas peticiones requerían la posibilidad de ser configuradas por el usuario. Las diferencias se pueden observar comparando la *Figura 3.6* contra la *Figura 3.50* ubicada en la siguiente página.

En la configuración de la captura establecí que para un canal en específico los valores a modificar son: el intervalo de grabación de cada segmento, la ubicación donde se deben de grabar los archivos, habilitar el cambio de segmento a la hora exacta del reloj (por ejemplo: las 3 P.M., 4 P.M., etc.), y sus reglas de mantenimientos como el espacio mínimo en disco de captura que se debe de tener en todo momento así como el tiempo de vida de los archivos con su unidad en días.

Dentro de los ajustes del almacenamiento masivo (*Storage*) están ubicados los valores de: el directorio al cual el usuario desea que se copien los archivos después de ser capturados, si se desea que se borren del directorio de captura después de ser copiados, y de la misma manera que en la captura, el tiempo de vida de los archivos y el espacio mínimo en disco que debe estar libre en todo momento.

En los ajustes de audio coloqué la configuración de la codificación del audio, que en el caso del hardware utilizado durante este desarrollo solamente ofrece la codificación AAC. El muestreo del audio al momento de digitalizarlo de 22.05, 32, 44.1 y 48 KHz. En esta categoría también el usuario puede configurar la ganancia de cada canal de audio por separado con valores desde -96 hasta 12 dB. Solo el audio puede ser configurado dinámicamente, las otras dos opciones requieren que la captura se reinicie.

Con respecto a la configuración del video, coloqué controles que permiten elegir la entrada del video de la cual se quiere capturar en la entrada, donde sus posibles valores son: entrada compuesta o S-Video. También se encuentra otro control que establece el formato ya sea NTSC o PAL; junto con otro control donde el usuario puede establecer la resolución de video, los posibles valores de este control están descritos en la Tabla 3.2 - Resoluciones de video soportadas por la tarjeta MM400 (pág. 38). Debido a que la tarjeta capturadora tanto el brillo, contraste, coloración, saturación y bitrate pueden ser modificados dinámicamente durante la captura.

Figura 3.50 - Diálogo final de configuración

3.6 Pruebas al Sistema

Una prueba importante que tuve que realizar al sistema era la verificación del manejo correcto de la memoria de cada módulo, con la finalidad de asegurar la estabilidad del software. Esta prueba consistía en ejecutar el módulo *Server*, posteriormente el módulo *Client* y comencé a grabar en todas las entradas disponibles.

Todos los canales tenían: el mismo tiempo de cambio de segmento, el traslado de archivos al almacenamiento masivo activado y todas las opciones de mantenimiento habilitadas.

Una vez que configuré apropiadamente el ambiente de trabajo monitoreaba los siguientes parámetros de memoria del proceso:

- Tamaño de la memoria utilizada
- Tamaño de la memoria virtual
- Handles
- Objetos GDI

El monitoreo de estos valores lo pude llevar a cabo mediante la aplicación *Fission Memory Check* que desarrollé con este propósito. Mediante dicha aplicación podía obtener los valores a analizar mediante las llamadas al sistema como: `GetProcessMemoryInfo`, `GetProcessHandleCount` y `GetGuiResources` en periodos predeterminados. Las pruebas expuestas en este escrito las ejecuté en base a mediciones realizadas cada minuto durante 24 horas.

En esta prueba buscaba obtener un panorama general del comportamiento del uso de la memoria por los módulos para localizar fugas de memoria (*memory leak*). Si la memoria crecía en todo momento, sin ser liberada, era indicio de que existía este error.

Si encontraba esta falla, era necesario que utilizara un programa más avanzado que en ocasiones me auxiliaba a ubicar el error que originaba la falla dentro del código, como lo es el programa *Memory Validator* de la compañía *Object Media Limited*.

Desafortunadamente durante las pruebas continuas que hice durante el desarrollo aprendí que existen ocasiones en donde la reparación de las fugas de memoria no estaba en mi alcance.

Una muestra particular del desarrollo es cuando está ubicada la falla en las bibliotecas dinámicas que ofrece el SDK de Optibase. Como no contaba con el código fuente del SDK para ser capaz de corregir directamente el problema, me enfrenté a dos posibles soluciones. La primera era solicitarle a Optibase la corrección de dichos desperfectos en su paquete de desarrollo. No obstante, no todos los errores eran corregidos, por lo tanto, tuve que ingeniar artificios dentro del software, como reinicios de la aplicación, para que se libere la mayor cantidad de memoria posible, sin embargo esto deteriora la memoria del sistema a largo plazo.

En cuanto a los handles que inspeccioné, es importante comprender que dentro del núcleo (kernel) de Windows se almacena una tabla que contiene todos los objetos, de los cuales es responsable el sistema operativo. Esos objetos son las ventanas, los botones, los íconos, apuntadores del mouse, menús, etc., los cuales están referenciados en esa tabla y cada entrada dentro de la misma tabla tiene un identificador único llamado *handle*.

En términos más técnicos, un *handle* es un apuntador a un apuntador de ubicación de memoria que utiliza Windows internamente para llevar un control de los objetos en la memoria. Dado que Windows considera a los objetos como bloques de memoria, estos pueden llegar a ser reubicados para liberar espacio, obligando a la tabla de handles ser actualizada.

Una instancia de un programa que se ejecuta en Windows posee un límite teórico de 65536 handles de acuerdo a Microsoft. En caso de exceder ese límite el programa terminaría en un error fatal. Por este motivo decidí llevar un conteo de los mismos durante las pruebas.

En último lugar, pero no con menor importancia, los objetos GDI son handles de objetos gráficos utilizados por el programa y, consecuentemente, también tenía que contabilizarlos dentro de las pruebas para verificar que no crecieran desmesuradamente.

Debido a que el sistema se compone de 4 diferentes módulos, se realizaron pruebas independientes para cada uno de ellos.

3.6.1 Fission Monitor Server

En la siguiente gráfica se puede apreciar como la memoria se mantuvo estable durante un periodo de 24 horas. Las altas y bajas de memoria constantes son el reflejo de que los archivos están siendo capturados y que al cerrar el archivo, la memoria es liberada correctamente. Una particularidad de esta prueba hacia el final es que falló un cambio de segmento con lo que incrementó la carga de trabajo sobre este módulo sin embargo el sistema pudo corregir el fallo originado desde el SDK de Optibase.

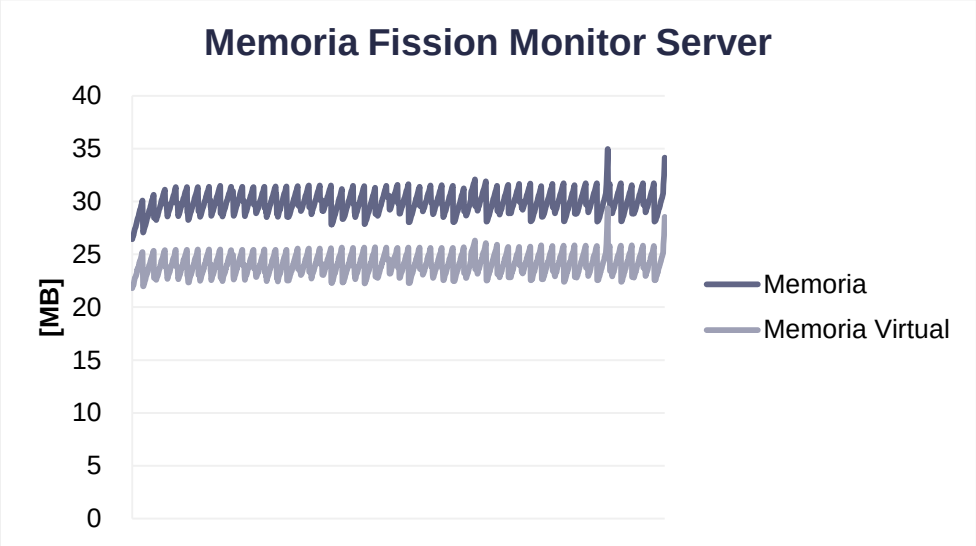


Figura 3.51 - Resultado de la prueba de memoria del Server

Los handles que utiliza el sistema se mantuvieron constantes en todo momento durante la prueba mostrando que en este aspecto es totalmente estable.

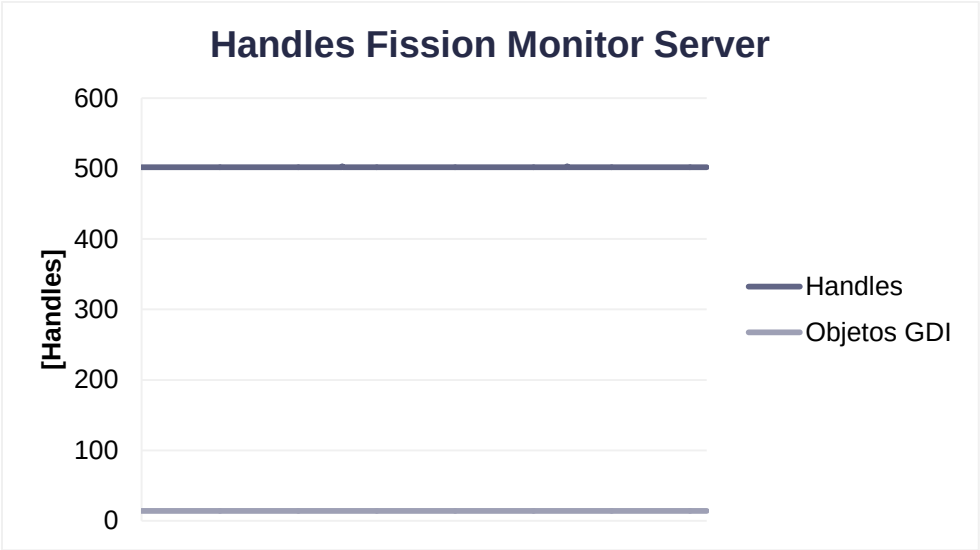


Figura 3.52 - Resultado de la prueba de handles del Server

3.6.2 Fission Monitor Upkeeper

Al igual que en el caso del servidor, la memoria se mantuvo estable durante la prueba. En esta gráfica también noté que el módulo Upkeeper tuvo un incremento en la carga de trabajo al tratar de recuperar el tiempo perdido por la falla del cambio de segmento originado en el servidor.

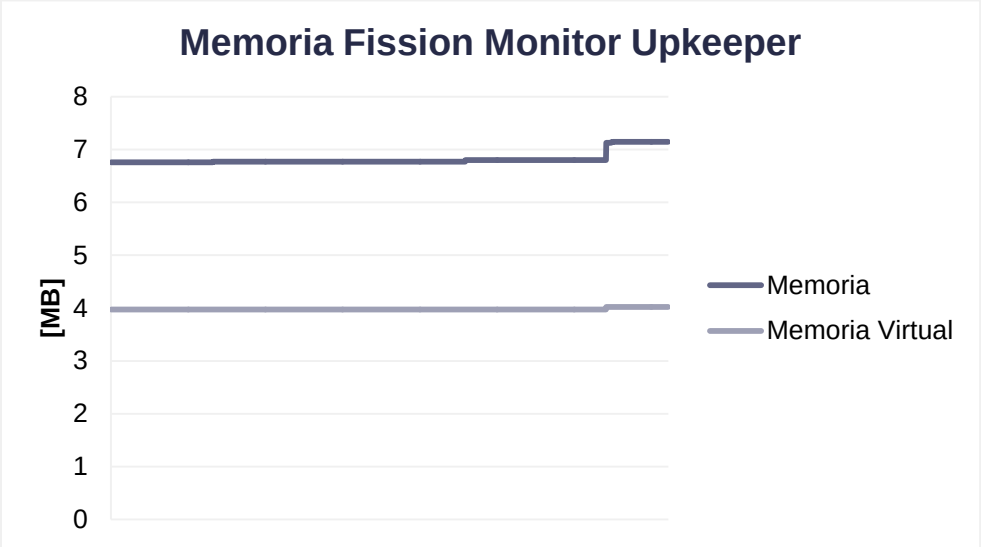


Figura 3.53 - Resultado de la prueba de memoria del Upkeeper

El conteo de *handles* refleja de la misma manera que el sistema incremento su carga de trabajo al aumentar en la misma proporción con respecto a la memoria utilizada antes del incidente.

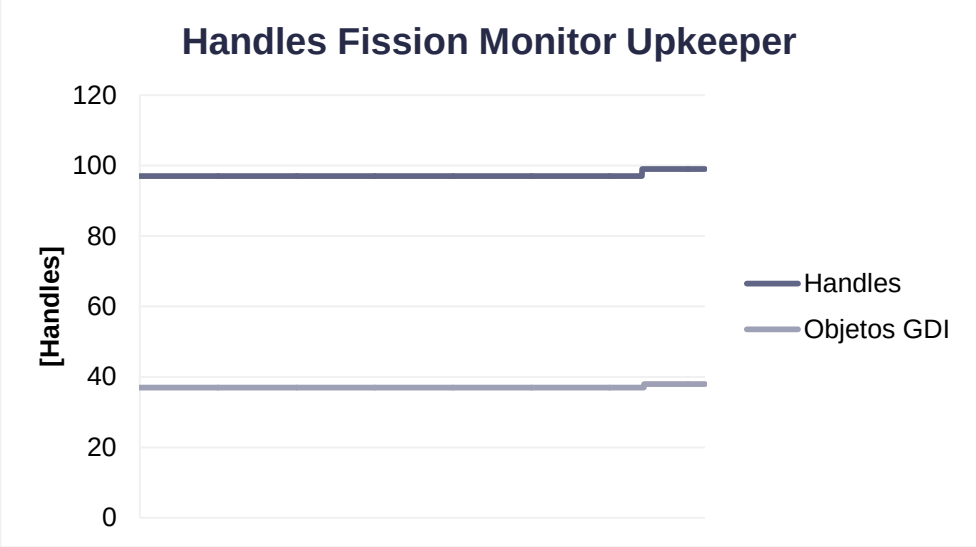


Figura 3.54 - Resultado de la prueba de handles del Upkeeper

3.6.3 Fission Monitor Client

Durante esta prueba el módulo se encontró desplegando el video de las cuatro entradas existentes dentro del sistema. De la misma forma que el servidor, mostró estabilidad dentro de esta prueba.

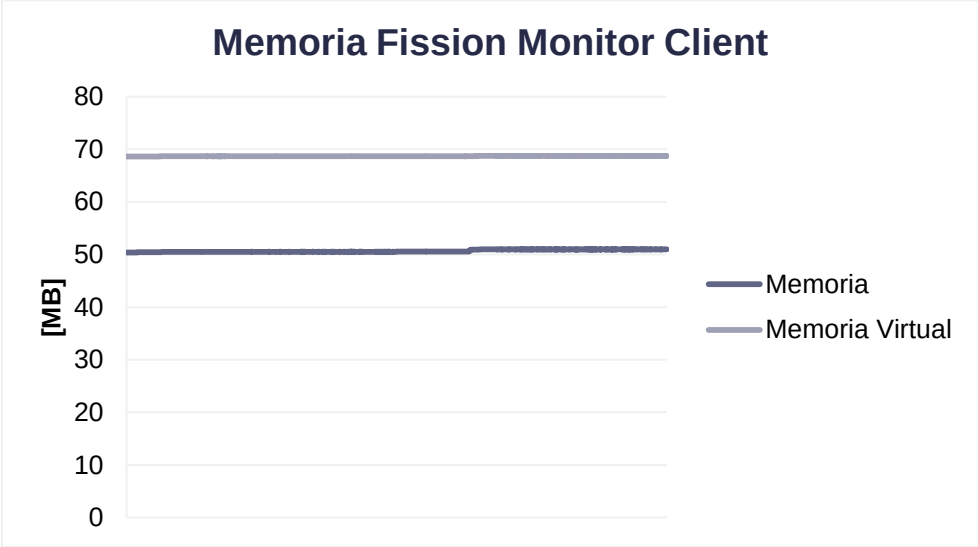


Figura 3.55 - Resultado de la prueba de memoria del Client

En cuanto a los *handles* se refiere, también observé que el comportamiento fue el deseado. Cuando comparé los *handles* que utiliza el módulo *Server* contra los que utiliza el módulo *Client*, pude percatarme como el módulo cliente posee más elementos del tipo gráfico mediante el conteo de objetos GDI debido a que el módulo *Client* brinda la interfaz gráfica del sistema.

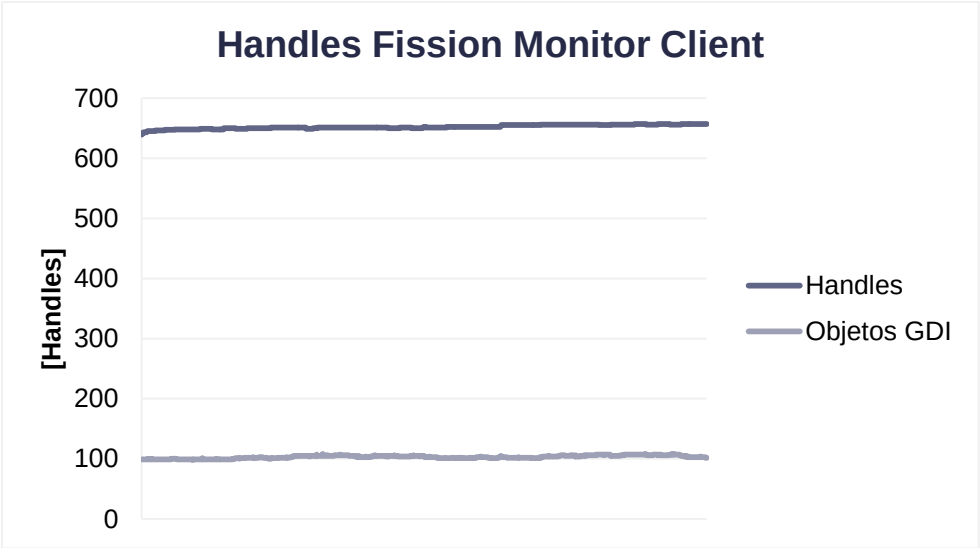


Figura 3.56 - Resultado de la prueba de handles del Client

Dado que el módulo *Client* ofrece una interfaz gráfica de todo el sistema, realiza búsquedas y reproduce video del registro de captura, fue necesario que efectuara otras pruebas aisladas donde se aprecie como se comporta la memoria durante la ejecución de estos procesos.

Pruebas al Visualizador

Para medir el rendimiento y estabilidad del visualizador del registro de captura, realicé una prueba que consistía en abrir y cerrar el visualizador. En una primera instancia, para comprobar que la memoria y los *handles* fueran liberados correctamente, realicé las mediciones pertinentes y obtuve los siguientes resultados:

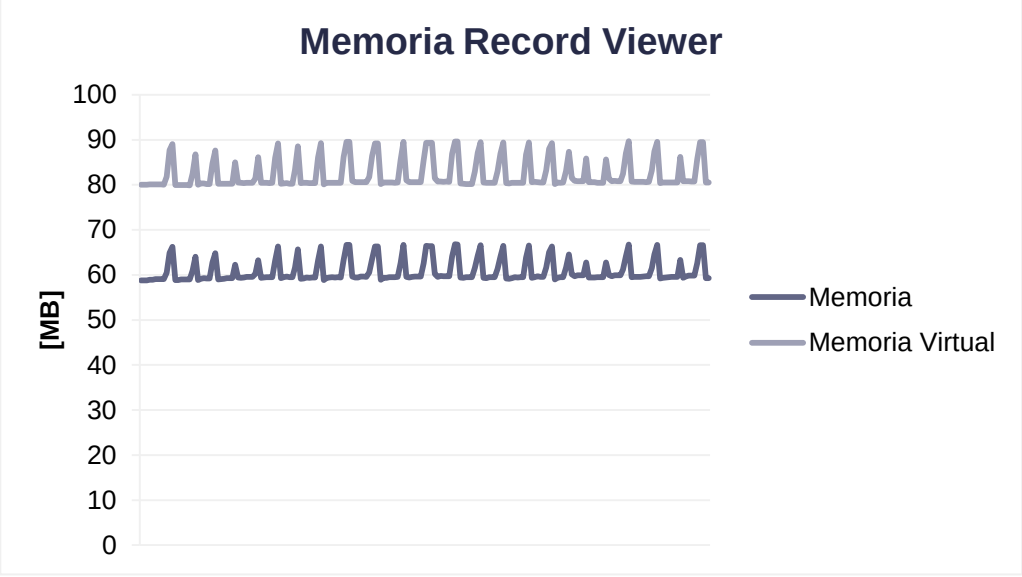


Figura 3.57 - Resultado de la prueba de memoria del Record Viewer

La liberación de memoria al cerrar el visualizador la califiqué como satisfactoria, ya que la memoria vuelve al nivel inicial al momento de cerrar el diálogo después de realizar una búsqueda.

Sin embargo, en cuanto a los *handles* pude ver que estos no los liberaba de la manera que deseaba. El número de *handles* aumentaba a una razón constante por cada ejecución del *Viewer* y por búsqueda como se muestra en la gráfica de los resultados de esta prueba.

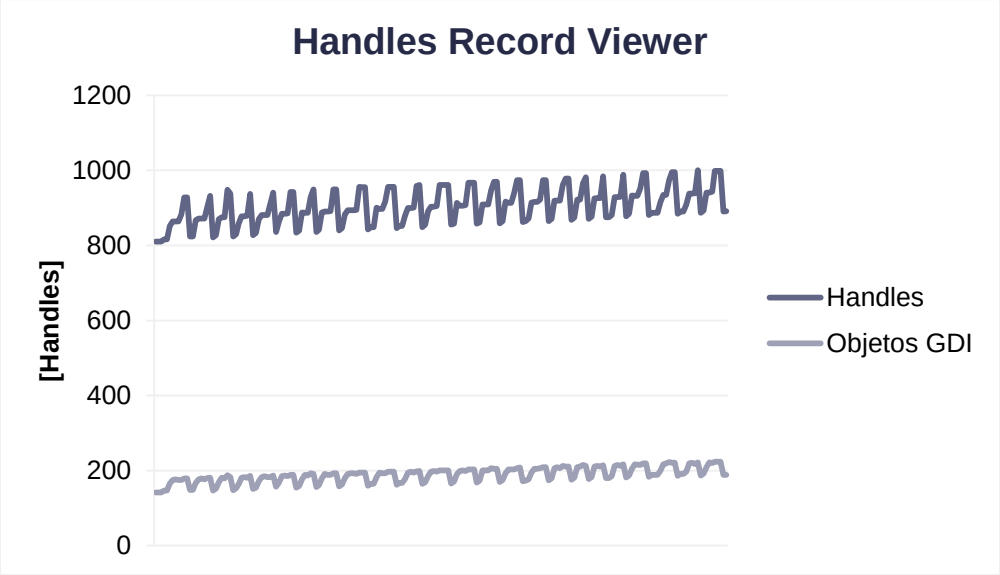


Figura 3.58 - Resultado de la prueba de handles del Record Viewer

Cuando observé la gráfica de los resultados me percaté de un incremento lineal y consideré que sería útil conocer un estimado del número de ejecuciones del visualizador que llevarían a la aplicación a un error fatal.

Para llegar a dicho cálculo obtuve los mínimos del conteo de *handles* que representan el estado del programa al haber cerrado el diálogo del visualizador. Los mínimos los obtuve mediante el cálculo de la primera y segunda derivada del número de *handles* con respecto al tiempo. Posteriormente, ubiqué los mínimos en las muestras donde la razón de cambio de *handles, o la primera derivada,* era nula y la segunda derivada es mayor a cero, en otras palabras, el cambio tenía una tendencia positiva.

Una vez que adquirí todos los mínimos, apliqué una regresión lineal sobre los datos, obteniendo una fórmula que me indicaría una aproximación del número de ejecuciones que causarían un error fatal de la aplicación, despejando la ecuación obtenida y considerando que el máximo número de *handles* permitidos por aplicación es 65536, conseguí que el número de ejecuciones que llevarían a la aplicación a un error fatal es de *~22398.*

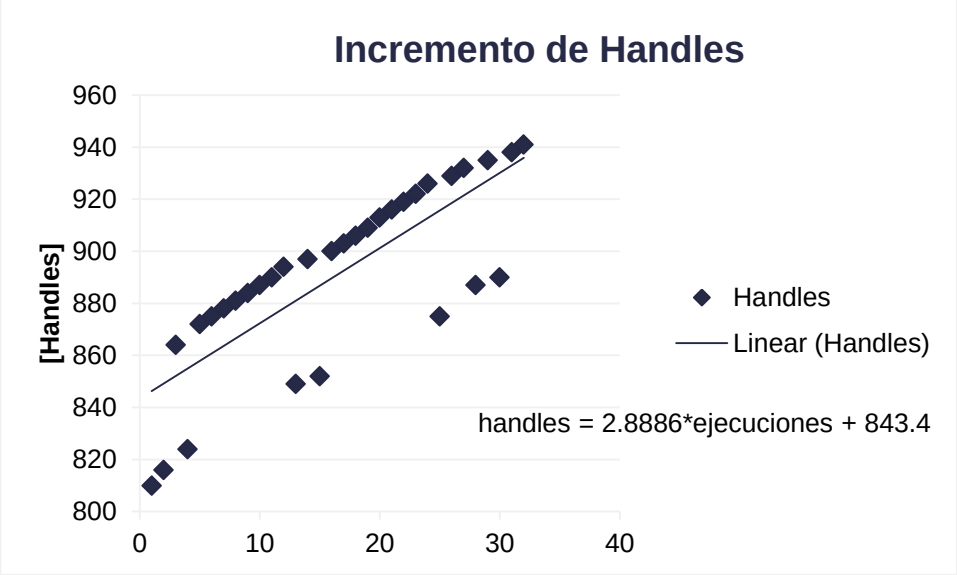


Figura 3.59 - Regresión lineal de los handles del Record Viewer

Consideré que era altamente improbable que un usuario llegara a ese número de búsquedas en una sola sesión y tomando ventaja de que Windows libera todos los *handles* utilizados por una aplicación de una manera automática al momento de cerrarla, concluí que este comportamiento no llevara al módulo *Client* a un error fatal en la mayoría de los casos de uso. Además, si esto llegara a ocurrir, la aplicación que fallaría sería el módulo que ofrece la interfaz gráfica, lo cual no afectaba la grabación.

No obstante, no deja de ser un error presente dentro del software y merece toda mi atención para repararlo. Opiné, que era muy probable, que el error estaba presente en la destrucción de la gráfica de reproducción de video creada mediante DirectShow, ya que los objetos GDI incrementaban proporcionalmente, aunque eso no descartaba la posibilidad de que el error estuviera presente en la destrucción inadecuada del diálogo del *Record Viewer*.

3.6.4 Fission Monitor Exporter

Las pruebas que realicé para el módulo de exportación de videoclips las hice sobre un registro que contenía aproximadamente 1200 archivos de 30 minutos cada uno, lo cuál da un registro aproximado de 25 días de grabación. Los archivos se ubicaban en 2 directorios distintos, estando la mayor parte de las grabaciones ubicadas en una NAS con una conexión de 1GB/s y el directorio de captura de archivo siendo un disco de 1TB con una velocidad de acceso a 7200 RPM. El procesador con el que se trabajó en la exportación por software era una Pentium D de dos núcleos a 2.8 GHz y contaba con 2 GB de memoria RAM.

En primer lugar hice pruebas para tomar el tiempo que tarda tanto la búsqueda como la exportación, con lo cual conseguí completar la siguiente tabla:

Tabla 3.6 - Resultado prueba de exportación

Longitud Videoclip [min]	Tiempo de Búsqueda [s]	Tiempo de Exportación [min]
10:00	0.359	03:00
20:00	0.312	06:01
30:00	0.624	09:14
40:00	0.324	12:24
50:00	0.296	15:26
60:00	0.639	18:46

En los resultados de esta tabla pude observar que el tiempo de exportación es relativamente lineal y se puede realizar en menos de la mitad del tiempo de la duración del videoclip a exportar. Este es un factor importante dentro del punto de venta ya que el usuario espera que como máximo la exportación sea en una velocidad de 2X con respecto a la longitud del clip.

Los tiempos de búsqueda, menores a un segundo, los encontré totalmente satisfactorios, ya que en la mayoría de las estaciones de televisión almacenan un mes de grabación en un escenario similar.

Con respecto al manejo de memoria durante el uso de los servicios de exportación de multimedia que ofrece Microsoft, encontré que la memoria utilizada por la aplicación incrementa a razón de aproximadamente 5 MB por cada segmento de diferente archivo incluyendo el video en negros que se expuso como Caso 5 en la sección 6.4.5.

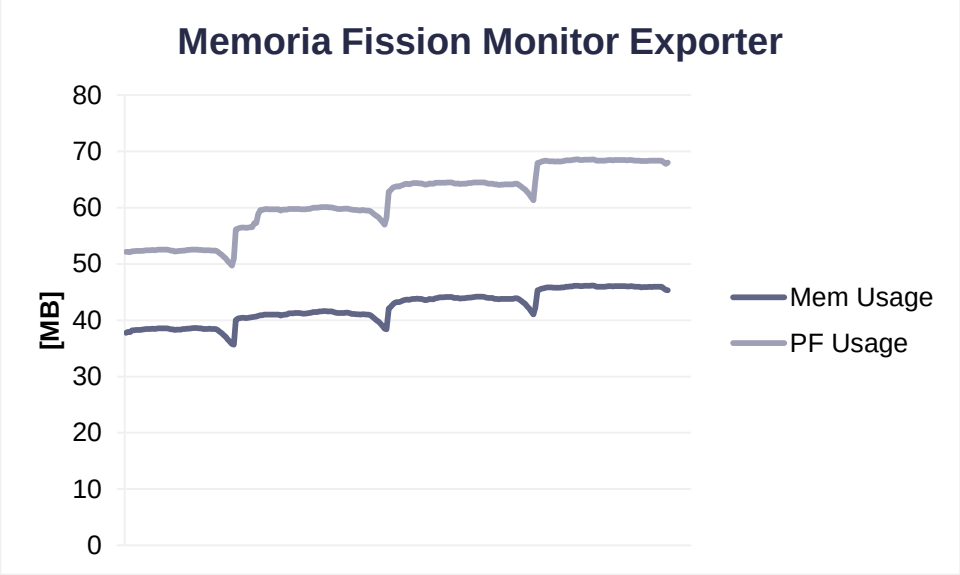


Figura 3.60 - Resultado de la prueba de memoria del Exporter

También observé el mismo comportamiento con los *handles* utilizados..

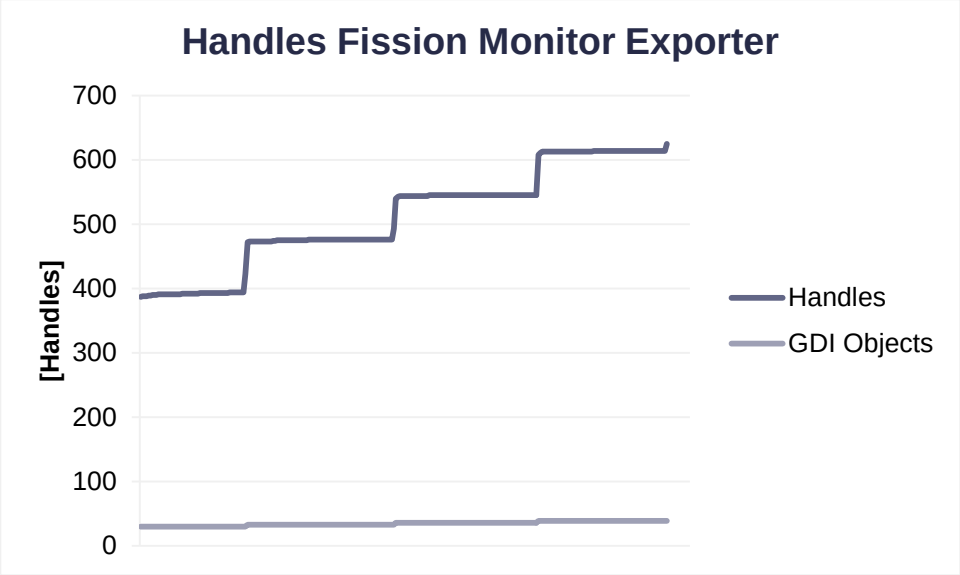


Figura 3.61 - Resultado de la prueba handles del Exporter

Por lo tanto al momento de exportar, me resultó adecuado colocar un cálculo para verificar si el ordenador que va a realizar la exportación cuenta con los recursos necesarios de memoria disponibles para realizar una exportación donde la memoria requerida por exportación es igual a:

$$M_e = S * K_e$$

Donde:

M_e es la memoria requerida para exportar

S es el número de segmentos durante la exportación

K_e es la constante de memoria requerida por los Microsoft Exporting Services en cada segmento

De una manera similar calculé la cantidad de *handles* requeridos por la exportación, con la diferencia de que la constante en este caso son el número de *handles* utilizados por segmento.

Dentro del *Client,* una vez calculada la memoria y los *handles* requeridos para la exportación era necesario compararlos con su contraparte disponible en el sistema operativo. Si en esta comparación el programa determinaba si existían los recursos suficientes, la exportación se llevaba a cabo. En caso contrario, se le notificaría al usuario final un mensaje de error durante la exportación.

3.7 Resultados del Proyecto

La versatilidad del sistema se mostró mediante el uso adecuado de las interfaces proporcionadas por DirectShow. Éstas permitieron que el producto no estuviera ligado a un códec de video en particular, permitiéndole al cliente instalar los códecs de video que considere más adecuados a su presupuesto y necesidades, obteniendo un beneficio con respecto a los costos de licenciamiento.

El desarrollo modular que utilicé durante el diseño e implementación del sistema me permitió generar un producto que ofrece una mejora continua. La función principal de grabar las señales de audio y video fue interrumpida en pocas ocasiones, gracias a que todos los módulos del *Fission On-Air Monitor*, salvo el módulo Server, podían ser actualizados sin la necesidad de detener la grabación.

El modelo C/S mostró ser de gran utilidad para la implementación del sistema, principalmente por dos razones: la primer, ayudó a desarrollar una versión base del producto que era funcional en un lapso muy breve, ya que gran parte de la implementación fue concentrada fuera del módulo Monitor Server; la otra razón, es que ayudó a minimizar la interrupción de la grabación por fallos lógicos de la programación. En un inicio, estos fallos se presentaron durante las implementaciones de la búsqueda o de la visualización de los archivos capturados. Así logré el cometido de que si ocurría un error fatal fuera del módulo Server, la grabación no era afectada.

Como en todo producto de software, las mejoras siempre están presentes y este caso no es la excepción. Una mejora significativa dentro del sistema sería implementar una búsqueda más rápida y eficiente, mediante el uso de una base de datos. Dentro de la base de datos se registrarían las propiedades de audio y video de los archivos capturados, así como las posibles ubicaciones donde se pudiera encontrar el archivo correspondiente. Opino que este cambio volvería más eficiente la búsqueda, ya que la información sobre los archivos se tendría a priori, eliminando el tiempo necesario para abrir un archivo para extraer dicha información, y así se simplificaría la búsqueda a una consulta en la base de datos y una comparación de los datos almacenados en la base de datos contra los datos reales del archivo resultante de la consulta. Consulta que estaría estructurada para ubicar el archivo multimedia que contenga el video relacionado con la fecha y hora de la búsqueda.

La base de datos podría ser remota, en un ordenador externo, o local, dentro del mismo ordenador que alberga el hardware de captura Con esta modificación el tiempo de procesamiento n.O(n) de una búsqueda lineal, se reduciría de a k.O(n), donde n es el número de elementos a procesar y k sería el tiempo constante de acceso a la base de datos.

Otro aumento de las capacidades del sistema, sería ofrecer un mayor número de formatos de video al momento de exportar clips de la captura, ya que por el momento el software solo ofrece como opción el formato Windows ASF. El formato más solicitado

dentro de la gama de clientes de Fission Software, es el formato nativo de los archivos capturados, es decir, el formato MPEG-4 con el video codificado en H.263 y el audio en AAC. Sin embargo, esta modificación aumentaría el costo de los módulos *Client* y *Viewer*, o reduciría las ganancias si la compañía decidiera absorber el coste del licenciamiento por códec instalado.

Una característica del sistema que ha sido solicitada por diferentes clientes, es la capacidad de calendarizar las grabaciones en las distintas entradas de un clúster de servidores de grabación. Pese a que la problemática puede ser resuelta mediante la implementación de un servicio *(daemon)* centralizado en un equipo capaz de comunicarse con todos los servidores de grabación, puede resultar en un desastre de intercomunicación si los ordenadores y la red local no están propiamente configurados. Por lo tanto, la modificación más confiable sería agregar una base de datos en la red local que almacene los lapsos en los cuales las grabaciones deben ser realizadas y agregar dentro del módulo Server un proceso por entrada que verifique en todo momento, contra la base de datos, si ésta debe de estar grabando.

Por estos motivos, una modificación a considerar dentro del sistema es la adición de una base de datos.

Por el otro lado, considero que una mejora necesaria, de acuerdo a las tendencias del mercado, es ofrecer la funcionalidad que ofrece el módulo Client pero desde una estación remota al servidor de grabación a través de una red local o la Internet. Esto se podría lograr mediante diversas modificaciones al sistema, sin afectar drásticamente la base del sistema que establecí con este desarrollo.

Primero se debe de modificar el módulo Upkeeper, para que sea capaz de transferir los archivos capturados por medio de un protocolo de transferencia, como lo es el FTP, a un almacenamiento remoto. Una vez almacenados los archivos capturados en ese servidor, se debe implementar un servicio web que ofrezca el servicio de búsqueda y visualización de los videos en forma de aplicación web, a través de un navegador, como lo son el Mozilla Firefox o el Google Chrome. Afortunadamente, hoy en día la tecnología HTML5 permite la visualización de archivos de video con audio a través de la Internet de una manera nativa, siempre y cuando los archivos multimedia se encuentren en los formatos: MPEG-4, OGG o WebM.

En caso de elegir el formato MPEG-4, el video debe de estar codificado en H.264 y el audio en AAC, la desventaja de este formato es que el costo de los codecs es relativamente alto lo cual impactaría en el precio de la solución. No obstante, resulta más fácil conseguir hardware que codifique las entradas de A/V en dicho formato que en las otras dos opciones. Esto se debe a que este formato está obteniendo popularidad dentro del mercado del *broadcasting*.

OGG ha sido un formato con gran fama en la Internet, pero el códec Theora a pesar de tener un buen diseño base y ser de código abierto, no ha sido impulsado por la industria, por lo que su panorama indica que no podría competir contra el formato WebM que está siendo promocionado por Google.

WebM es un formato de código abierto y consiste de un video codificado en VP8 y el audio en Vorbis. Esta solución resulta atractiva ya que los codecs están licenciados bajo LGPL, siendo posible comercializar con ellos.

En una primera fase de esta mejora, se tendrían que transcodificar los archivos que graba la tarjeta Optibase a cualquiera de los formatos que se elija como opción y transferirlos a una unidad de almacenamiento que tenga acceso el servidor web. El orden de las dos últimas acciones es indistinto, al menos de que el ordenador que alberga el servidor web posea un hardware con mayor capacidad de procesamiento.

En una siguiente etapa, sería conveniente contar con el hardware que capture el video nativamente en cualquiera de los formatos propuestos, para contar con la disponibilidad inmediata del material para ser transmitido remotamente.

Considero importante resaltar que el desarrollo de la aplicación web a la que tiene acceso el usuario final tenga implementado al menos la funcionalidad con la que cuenta el módulo *Fission Monitor Client* actualmente.

Esta es una razón de peso que vuelve atractiva la idea de implementar dichos servicios sobre la plataforma Windows Server, ya que el código que implementé debe de ser fácil de portar a la tecnología web que ofrece Microsoft. No obstante, no se debe de descartar el desarrollo sobre un servidor con Linux, ya que ésta es una plataforma popular para servicios web, hoy en día.

Conclusiones

Para concluir, durante mi estancia en Fission Software he aprendido que el desarrollo de software es muy importante en países como México, ya que por medio de éste se pueden obtener resultados de alta tecnología a un bajo costo, mediante el remplazo de equipos especializados de un alto costo por medio de un ordenador que alberga un software, el cuál tiene su costo relevante son las horas hombre empleadas para su desarrollo.

También aprendí que el desarrollo de software puede llegar a ser una actividad complicada de cuantificar y controlar debido a la abstracción inherente que ésta conlleva. Una enseñanza que adquirí durante mi trabajo es que todo desarrollador requiere de una comunicación detallada con los clientes para obtener una comprensión correcta de lo que solicita y también necesita un canal abierto con el resto de los departamentos dentro de la empresa para acelerar los resultados.

Como acabo de mencionar, considerando que durante el desarrollo de software uno de los costos más importantes durante el desarrollo de software son las horas invertidas, resulta indispensable realizar un análisis previo de una manera minuciosa de la problemática a resolver.

El fin de esta tarea es obtener una dirección apropiada en el desarrollo, ya que de otra forma se podría llegar al punto donde el desarrollo se enfrena al llegar a un resultado no deseado, desperdiciando tiempo en ese desarrollo erróneo, obligando a comenzar un nuevo desarrollo para lograr el resultado deseado por el cliente. Sin embargo, ese tiempo puede tener fuertes repercusiones como: penalizaciones dentro del pago, en el mejor de los casos, hasta la pérdida de un cliente.

Reiterando, el análisis previo es una tarea indispensable dentro del desarrollo de software aplicado en cualquier área para obtener resultados sin contratiempos.

En otro aspecto también aprendí que las bases matemáticas son indispensables para obtener un ágil desarrollo y un producto escalable ya que con técnicas de recursividad y abstracción de tipos se puede desarrollar código basado en una estructura genérica que puede ser utilizado en distintos programas apoyando al ágil desarrollo de las aplicaciones solicitadas por los clientes.

Durante mi estancia laboral en los últimos años tuve la oportunidad de aplicar conocimientos básicos adquiridos durante mi estancia en la Facultad de Ingeniería, pero sobre todo, los profesores inculcaron dentro de mi formación profesional el ser auto-didacta y buscar una mejora continua en mi formación profesional.

Apéndice A

Manejo de GraphEdit

Abrir un archivo usando *Intelligent Connect*

GraphEdit es una herramienta que está incluida en el SDK de DirectShow proporcionado por Microsoft. Con dicha herramienta se pueden construir y probar diversas gráficas de filtros para DirectShow de una forma visual. Los filtros pueden ser vistos como rectángulos con su nombre en el área interna. Los pines de cada filtro aparecen en las orillas como pequeños rectángulos sobre las caras laterales. Los pines de entrada se encuentran ubicados en el lado izquierdo del filtro y los pines de salida en el lado derecho del mismo. Las conexiones entre pines son representadas como flechas con dirección del pin de salida de un filtro hacia el pin de entrada de otro filtro.

GraphEdit mostró ser una herramienta útil para el proyecto en discusión debido a que permite construir una gráfica de reproducción para cualquier archivo de video de una manera visual, arrojando el archivo desde el Explorador de Windows hacia el espacio en blanco dentro de la la aplicación. Con ella se puede averiguar, de una forma sencilla, los filtros decodificadores que utiliza DirectShow mediante la *Conexión Inteligente* para reproducir un archivo de video. Con esto se puede verificar que el sistema, en el cuál se instala el módulo Fission Monitor Viewer, se encuentre configurado con los filtros adecuados.

La tarea descrita anteriormente también se puede realizar por medio del uso de los menús. Primero se accede al menú “File”, posteriormente se selecciona la opción “Render Media File” y se selecciona el archivo a renderear.

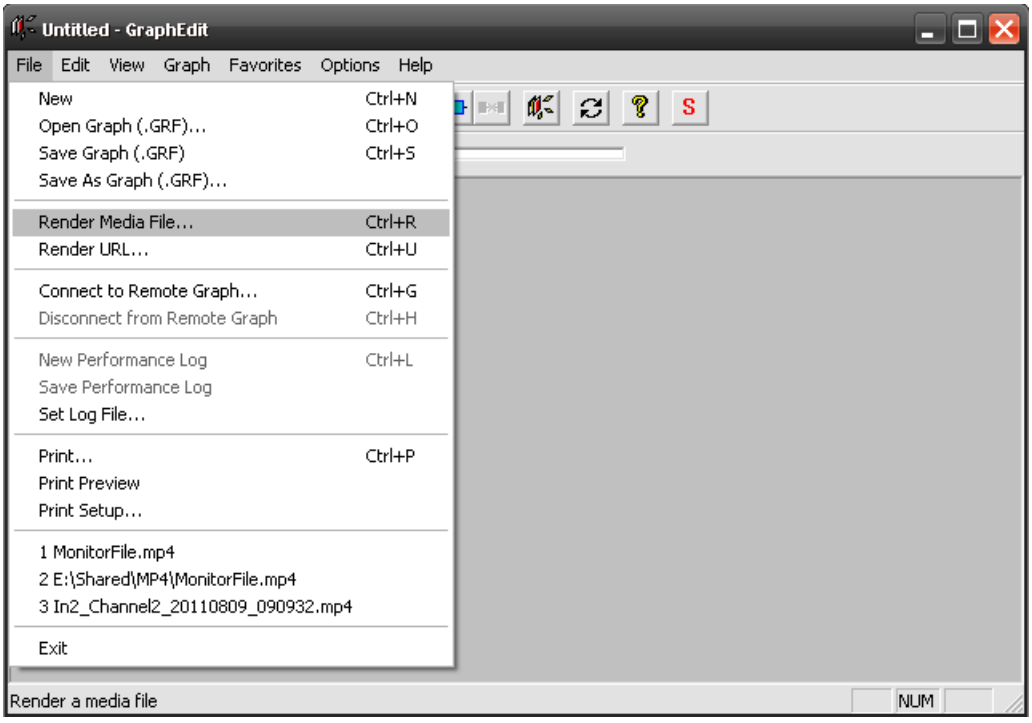


Figura A.1 - Abrir un archivo en GraphEdit por menús

Construcción manual de una gráfica

Para construir una gráfica de reproducción de un archivo de video con audio embebido, se debe de tomar en cuenta los filtros mínimos requeridos son:

- un filtro fuente
- un filtro demultiplexor de señales de audio y video
- un filtro decodificador de video
- un filtro decodificador de audio
- un filtro de salida de video
- un filtro de salida de audio.

Con fines didácticos, el ejemplo se realizará utilizando los filtros que proporciona el paquete de codecs K-Lite sobre un archivo con wrapper MP4 codificado en H.263 y los incluídos dentro del SDK de Directshow que proporciona Microsoft de manera gratuita:

Para constuir la gráfica hay que llevar a cabo los siguientes pasos dentro de un ordenador configurado con los filtros apropiados:

1. Ejecuta el programa *GraphEdit.*
2. Elegir le menú *Graph* y elegir *Insert Filters* o accesar por medio del atajo *Ctrl+F*.
3. Seleccionar *Directshow Filters* dentro de la lista de filtros, elegir el filtro ***File Source (Async.)*** y presionar el botón *Insert Filter*
4. Elegir el archivo a ser reproducido por medio del menú de selección de archivo.
5. Agregar el demultiplexor ***MPC - MP4 Splitter*** de la misma forma en la que se agregó el filtro fuente en los pasos 1 al 3
6. Conectar la salida *Output* del filtro fuente a la entrada Input del filtro *MP4 Splitter*.
7. Agregar el decodificador de video ***ffdshow Video Decoder***
8. Conectar la salida *video(eng)* del demultiplexor a la entrada *In* del decodificador de video
9. Agregar el decodificador de audio ***ffdshow Audio Decoder***
10. Conectar la salida *audio(eng)* del demultiplexor a la entrada *In* del decodificador de audio
11. Agregar el filtro ***Video Renderer***
12. Conectar la salida *Out* del filtro decodificador de video a la entrada *Input* del filtro de salida de video
13. Agregar el filtro ***Default DirectSound Device*** ubicado bajo la categoría de *Audio Renderers*
14. Conectar la salida Out del filtro decodificador de audio a la entrada Audio Input Pin del filtro de salida de audio.

Una vez efectuados estos pasos, ya se cuenta con una gráfica simple de reproducción de video de los archivos capturados por el módulo Fission Monitor Server, el resultado debe de ser similar a la gráfica que se muestra en la figura:

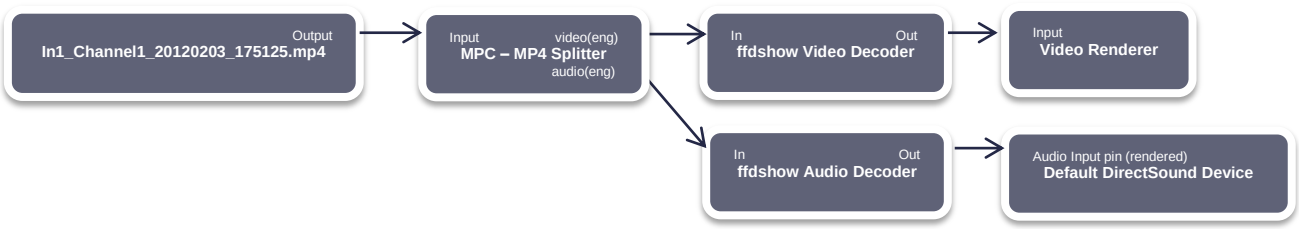


Figura B.1 - Gráfica de reproducción de video

Apéndice B

Filtro Fuente de una Gráfica

Dentro del módulo Fission Monitor Client implementé un reproductor de video, para que el usuario final pudiera visualizar los archivos dentro de los registros de captura, ya sea en el almacenamiento local o en la unidad de almacenamiento masivo, dependiendo de la lista resultante de la búsqueda de la fecha y hora solicitada.

Durante la reproducción de la lista de archivos, el reproductor se detenía al momento de terminar de reproducir un segmento (o archivo), destruía totalmente la gráfica que reproducía el video y creaba una nueva para reproducir el archivo secuencial de la lista mediante el uso de la *Conexión Inteligente* de DirectShow.

Durante una de las iteraciones del producto, el cliente solicitó que el reproductor de video disminuyera el tiempo de cambio entre segmentos del registro, ya que esto entorpecía el trabajo de los usuarios,

Para agilizar el cambio entre archivos, decidí programar un cambio dinámico del filtro fuente de la gráfica de reproducción. Con esta acción la tecnología de *Conexión Inteligente* no tenía que crear nuevamente los demultiplexores, decodificadores y filtros de salida que fueron creados al momento de realizar la primera llamada a *RenderFile*.

Sin embargo, esta implementación no es trivial, debido que una de las desventajas de crear la gráfica por medio de la llamada a *RenderFile* del objeto COM: *IGraphBuilder,* es que no se posee un apuntador o referencia al filtro fuente de la gráfica, por lo que es necesario implementar la búsqueda y eliminación del filtro fuente llamando al método programado *RemoveSourceFilter*, con lo que a continuación se puede agregar un filtro fuente con el siguiente archivo y volver a llamar a *RenderFile*.

```
bool RemoveSourceFilter(IGraphBuilder*  pGB)
{
   IEnumFilters      *pEnumFilters = NULL;
   IBaseFilter       *pFilter;
   IEnumPins         *pEnumPins = NULL;
   IPin              *pPin;
   PIN_DIRECTION pinDir;
   int        inPins, outPins;
   ULONG     cFetched;

   HRESULT    hr = pGB->EnumFilters( &pEnumFilters );
   if(FAILED(hr))
        return false;
   while( pEnumFilters->Next(1, &pFilter, &cFetched) == S_OK)
   {
        hr = pFilter->EnumPins( &pEnumPins );
        if(FAILED(hr))
              return false;
        // Cuenta los números de pins que posee el filtro
        outPins = 0; inPins = 0;
        while( pEnumPins->Next(1, &pPin, &cFetched) == S_OK)
        {
              pPin->QueryDirection( &pinDir );
              if(pinDir == PINDIR_OUTPUT)
                    outPins++;
```

```
                else
                        inPins++;
        }
        // Si el filtro no tiene entradas, es el filtro fuente
        if(inPins == 0)
        {
                IPin    *otherPin;
                pPin->ConnectedTo(&otherPin);
                pGB->Disconnect(otherPin);
                pGB->Disconnect(pPin);
                pPin->Release();
                otherPin->Release();
                pEnumPins->Release();
                pFilter->Release();
                hr = pGB->RemoveFilter(pFilter);
                if(FAILED(hr))
                        return false;

                break;
        }
        pPin->Release();
        pEnumPins->Release();
    }

    pEnumFilters->Release();

    return true;
}
```

Cabe la pena resaltar, que dentro de este método de búsqueda, se asume que el filtro fuente es el único filtro que no posee un pin de entrada y que el software se encuentra instalado en un ambiente que la compañía valide como compatible con el producto.

En caso de que se utlizara un codec sin validación por parte de la compañía es probable que el software falle al momento de intentar reproducir el video, por lo que resulta conveniente poner suma atención en este punto, ya que es probable que sea el origen de la falla en el producto.

Apéndice C

Protocolo de Comunicación

Antes de comenzar a describir los comandos que acepta el servidor, es importante recalcar que algunos de los resultados son cadenas de caracteres en formato XML definidos arbitrariamente para presentar la información de una manera ordenada y comprensible para un programa ajeno al sistema que desarrollé. De esta forma fue posible transmitir una estructura de datos completa por medio de una sola cadena de caracteres.

Dichas estructuras están definidas por medio de un DTD, de acuerdo a Erik Ray[1]. la definición de tipos de documento (DTD), tiene el propósito de facilitar el diseño de los mensajes y servir como una herramienta para comunicar a otros desarrolladores la forma en que se está formateando el mensaje. Dichas DTD's deben de encontrarse bien documentadas mediante comentarios para poder contextualizar los posibles valores que un campo pueda tener.

En este apéndice se da un listado de todos los comandos que son aceptados por el servidor a la fecha 4 de Febrero del 2012

Inicialización de la entrada

load <num_entrada>

Descripción Establece la entrada con los valores de configuración establecidos en el registro. *(Este comando ya no se usa dado que el comando init realiza esta tarea implícitamente)*

Parámetros <*num_entrada*>

Ejemplo
```
>load 2
ok
```

---

init <num_entrada>

Descripción Inicia una sesión de visualización con los ajustes de audio/video que se encuentran en el registro de la entrada correspondiente.

Parámetros <*num_entrada*>

Ejemplo
```
>init 0
ok
```

---

1. [1] Ray, Erik T., **Learning XML**, 2a ed., O'Reilly Media, 2003, E.U.A, págs. 113-119

Manipulación de la captura

start <num_entrada>

Descripción Inicia una sesión de captura con los ajustes de audio/video de la entrada correspondiente que se encuentra en el registro

Parámetros <*num_entrada*>

Ejemplo
```
>start 3
ok
```

---

stop <num_entrada>

Descripción Detiene la sesión que se encuentra en progreso, ya sea de captura o de visualización

Parámetros <*num_entrada*>

Ejemplo **`>stop 3`** (*Detiene la sesión en progreso de la cuarta entrada*)
```
stop C:\Captures\Input1.sdp
```

---

term <num_entrada>

Descripción Finaliza totalmente la entrada indicada

Parámetros <num_entrada>

Ejemplo **`>term 0`** (Cierra la primer entrada)
```
ok
```

---

reinit <num_entrada>

Descripción Reinicia la sesión que se encuentra corriendo actualmente. Detiene totalmente la sesión de visualización o captura y la vuelve a inciar

Parámetros <*num_entrada*>

Ejemplo **`>reinit 5`** (*Reinicia la sesión de la sexta entrada*)
```
ok
```

---

cut <num_entrada>

Descripción Segmenta la captura en un nuevo archivo de la entrada indicada

Parámetros <*num_entrada*>

Ejemplo **`>cut 3`** (*Segmenta la captura de la cuarta entrada*)
```
cut C:\Captures\Input1\In1_Channel1_20111124_234501.mp4
```

Configuración del sistema

set <num_entrada> <configuración> <valor>

Descripción Establece la configuración descrita dentro de los parámetros al valor que se indique, cabe resaltar que el indicador de configuración debe de estar precedido por un guión para ser reconocido

Parámetros <*num_entrada*> -<*configuración*> <*valor*>
Donde <*valor*> depende de <*configuración*>, a continuación se muestra una lista de todos los descriptores reconocidos para <*configuración*>

-path

Descripción Establece la ruta en la que se van a capturar los archivos de la entrada indicada

Valor <*dirección*>

Ejemplo **`>set 1 -path C:\Capturas\`** (*Guarda "C:\Capturas" como la dirección donde se capturan los archivos de video de la segunda entrada*)

-stenable

Descripción Habilita o deshabilita si los archivos capturados se van a mover a un almacenamiento externo.

Valor [0 | 1] 0: deshabilita, 1:habilita

Ejemplo **`>set 2 -stenable 0`** (*Deshabilita el almacenamiento masivo de la tercera entrada*)

-stpath

Descripción Asigna la ruta del almacenamiento externo a la cual deben ser transferidos los archivos capturados

Valor <*dirección*>

Ejemplo **`>set 1 -stpath J:\Capture 1\`** (*Salva la dirección "J:\Capture 1" como el directorio donde se guardan los archivos de A/V capturados*)

-stmove

Descripción Establece si los archivos se van a borrar del directorio de captura una vez copiados en el almacenamiento externo.

Valor [0 | 1] (0: deshabilita, 1:habilita)

Ejemplo **`>set 2 -stmove 0`** (*Solamente copia los archivos al almacenamiento masivo*)

-name

Descripción Asigna el nombre (valor alfanumérico) a la entrada correspondiente.

Valor <*nombre*>

Ejemplo **`>set 2 –name Canal2`** *(Asigna el nombre Canal2 a la tercer entrada)*

-interval

Descripción Fija la duración del cambio de segmento, las unidades de la duración es en milisegundos.

Valor <*duración*>

Ejemplo **`>set 1 –interval 360000`** *(Establece que el cambio de segmento ocurra cada hora 1000[ms/s]*60[s/min]*60[min/hr] = 360000[ms])*

-mode

Descripción Configura el formato en que se va a grabar la señal de video.

Valor 1: NTSC, 2: PAL

Ejemplo **`>set 0 -mode 1`** *(Configura la primera entrada a capturar en formato NTSC)*

-bri

Descripción Configura dinámicamente el brillo del video.

Valor [0, 256]

Ejemplo **`>set 3 -bri 128`** *(Configura el brillo a un valor medio de 128 en la cuarta entrada)*

-con

Descripción Configura dinámicamente el contraste del video.

Valor [0, 256]

Ejemplo **`>set 3 -con 200`** *(Configura el contraste a un valor medio de 128 en la cuarta entrada)*

-hue

Descripción Configura dinámicamente el matiz del video

Valor [0, 256]

Ejemplo **`>set 3 -hue 128`** *(Configura el matiz a un valor medio de 128 en la cuarta entrada)*

-sat

Descripción Configura dinámicamente la saturación del video

Valor [0, 256]

Ejemplo **`>set 3 -sat 128`** *(Configura la saturación a un valor medio de 128 en la cuarta entrada)*

-source

Descripción Configura el tipo de entrada de video a utilizar

Valor [1 | 2]1: Compuesto 2: S-Video

Ejemplo **`>set 3 -source 1`** *(Establece la entrada compuesta como la activa de la cuarta entrada)*

-vbrate

Descripción Establece la tasa de bits en la que se quiere capturar el video, la unidad es en bits por segundo [bps].

Valor Depende de la resolución de video:

Resolución		**Bitrate**
NTSC	**PAL**	
160x112	192x144	8 Kbps – 380 Kbps
176x112	176x144	
320x240	352x288	320 Kbps – 3 Mbps
352x240	352x288	

Ejemplo **`>set 1 –vbrate 512000`** *(Configura la tasa de bits a 512 Kbps para la segunda entrada)*

-vres

Descripción Establece la resolución de video en la que se va a grabar la entrada, el valor de la resolución es un entero como se describe en la siguiente fórmula.

Valor *<resolución>* = *<ancho>* << 16 + *<alto>*
Acarrea 16 bits a la izquierda la altura y se suma el ancho de la resolución.

Ejemplo **`>set 0 –vres 20971760`** *(Configura la resolución de la primer entrada a 320x240)*

-aenable

Descripción Habilita o deshabilita el audio de la entrada.

Valor [0 | 1] (0: deshabilita, 1:habilita)

Ejemplo **`>set 3 –aenable 1`** *(Habilita la captura del audio en la cuarta entrada)*

-asampling

Descripción Configura la frecuencia de muestreo del audio capturado, la unidad está en KHz.

Valor [11000 | 22000 | 32000 | 44000 | 48000]

Ejemplo **`>set 0 –asampling 44000`** *(Configura el muestreo a 44 KHz)*

-again

Descripción Configura la ganancia del audio de ambos canales, la unidad está en [dB].

Valor [-96, 12]

Ejemplo **`>set 1 –again -45`** *(Configura la ganancia del audio capturado en la segunda entrada a -45 decibeles)*

-lgain

Descripción Configura la ganancia del canal izquierdo del audio, la unidad está en [dB].

Valor [-96, 12]

Ejemplo **`>set 1 –lgain 12`** *(Configura la ganancia del canal izquierdo del audio capturado en la segunda entrada a 12 decibeles)*

-rgain

Descripción Configura la ganancia del canal derecho del audio, la unidad está en [dB].

Valor [-96, 12]

Ejemplo **`>set 1 –rgain 9`** *(Configura la ganancia del canal derecho del audio capturado en la segunda entrada a 9 decibeles)*

Consultas al sistema

frate <num_entrada>

Descripción Consulta los cuadros por segundo a los que se captura el video en la entrada solicitada.

Parámetros *<num_entrada>*

Ejemplo **`>frate 3`** *(Obtiene los cuadros por segundo a los que captura la cuarta entrada)*
`15`

---

sdp <num_entrada>

Descripción Obtiene una cadena de caracteres que contiene el SDP para la visualización de la entrada.

Parámetros *<num_entrada>*

Ejemplo
> sdp 3 *(Solicita el SDP de la cuarta entrada)*
v=0
o=IOREK 509 544943936 IN IP4 10.0.1.96
s=MM-400
t=0 0
a=x-authentication:C50ED383269F761130BD172F
a=isma-compliance:1,1,1
a=mpeg4-iod: "data:application/mpeg4-iod;base64,AoI/AE8BAf/3AQOBUgABQLRkYXRhOmFwcGxpY2F0aW9uL21wZWc0LW9kLWF1O2Jhc2U2NCxBV2dCUndVZkEwTUE0QVFFT3lBUkFnZ0FBQStqSUFBUG95QUZMQUFBQWJEM0FBQUJzbFZWQUFBQnRRa0FBQUVCQUFBQklRQ080QUFpZ0tBQ0pBQlFRRWdBOVFzRUhoUmpCZ0VFQVIwQ253TVpBTUFGQkJGQUZRQUdBQUFBdTRBQUFBMdUFCUUlTRUFZQkF3PT0EDQEFAADIAAAAAAAAAAAGCQEAAAAAAAAAANhAAJAPmRhdGE6YXBwbGljYXRpb24vXBlZzQtYmlmcy1hdTtiYXNlNjQsd0JBU2dUQXFCWEpoQkloUVJRVS9BQT09BBICDQAAFAAAAAAAAAAABQMAAEAGCQEAAAAAAAAAA=="
m=video 11118 RTP/AVP 96
c=IN IP4 127.0.0.1
a=rtpmap:96 MP4V-ES/90000
a=fmtp:96 profile-level-id=247; config=000001B0F7000001B25555000001B5090000010100000121008EE0002280A002240050404800F50B041E1463
a=mpeg4-esid:224
m=audio 11138 RTP/AVP 97
c=IN IP4 127.0.0.1
a=rtpmap:97 mpeg4-generic/44100/2
a=fmtp:97 streamtype=5; profile-level-id=255; mode=AAC-hbr; config=1210; SizeLength=13; IndexLength=3; IndexDeltaLength=3; Profile=1
a=mpeg4-esid:192

---

totin

Descripción Consulta el número total de entradas existentes en el servidor.

Parámetros *N/A*

Ejemplo
>totin
4

---

ver

Descripción Solicita el número de versión del módulo Fission Monitor Server.

Parámetros *N/A*

Ejemplo
>ver
1.1.14.0 *(Esa es la versión del software)*

---

chlist

Descripción Responde con un listado indicando por cada entrada el nombre asignado, el formato de video, el estado en que se encuentra la entrada y la dirección ideal para el visualizador de contenido. La respuesta se encuentra en formato XML de acuerdo a la siguiente DTD:

```
<!--Channel List>
<!ELEMENT Channels (Input?)>
<!ELEMENT Input(Name, Mode, State, ViewerPath)>
<!ATTLIST Input
id #CDATA REQUIRED>
<!ELEMENT      Name    (#PCDATA)>
<!ELEMENT      Mode    (#PCDATA)>
<!ELEMENT      State   (#PCDATA)>
<!ELEMENT      ViewerPath (#PCDATA)>
```

Parámetros *N/A*

Ejemplo

```
>chlist
<Channels>
<Input id="0">
   <Name>Channel1</Name>
   <Mode>1</Mode>
   <State>0</State>
   <ViewerPath>D:\Captures\In1\</ViewerPath>
   </Input>
      <Input id="1">
      <Name>Channel2</Name>
      <Mode>1</Mode>
      <State>0</State>
      <ViewerPath>D:\Captures\In2\</ViewerPath>
   </Input>
   <Input id="2">
      <Name>Channel3</Name>
      <Mode>1</Mode>
      <State>2</State>
      <ViewerPath>D:\ Captures\In3\</ViewerPath>
   </Input>
   <Input id="3">
      <Name>Channel4</Name>
      <Mode>1</Mode>
      <State>0</State>
      <ViewerPath>D:\Captures\In4\</ViewerPath>
   </Input>
</Channels>
```

curset <num_entrada>

Descripción Consulta los ajustes de audio, video, captura y almacenamiento de la entrada indicada, el resultado son los datos en una cadena de caractéres en formato XML de acuerdo al siguiente DTD:

Parámetros *<num_entrada>*

Ejemplo

```
>curset 0 (Obtiene la configuración de la primera entrada)
<Settings>
   <General>
      <CaptureFolder>D:\Captures\In1\</CaptureFolder>
      <StorageEnabled>0</StorageEnabled>
      <StorageFolder>D:\Captures\</StorageFolder>
      <StorageMove>0</StorageMove>
      <ChannelName>Channel1</ChannelName>
      <Interval>1800000</Interval>
      <Mode>1</Mode>
   </General>
   <Video>
      <BRI>128</BRI>
      <CON>128</CON>
      <HUE>128</HUE>
      <SAT>128</SAT>
      <Source>1</Source>
      <VBR>1025000</VBR>
      <Resolution>23068912</Resolution>
   </Video>
   <Audio>
      <Enabled>1</Enabled>
      <Encoding>4</Encoding>
      <Sampling>44100</Sampling>
      <LGain>0</LGain>
      <RGain>0</RGain>
   </Audio>
   <Upkeep>
      <Capture>
      <Enabled>1</Enabled>
      <FileLifeSpan>0</FileLifeSpan>
      <FilesExpire>0</FilesExpire>
      <MinFreeSpace>46.50000</MinFreeSpace>
   </Capture>
   <Storage>
      <Enabled>0</Enabled>
      <FileLifeSpan>0</FileLifeSpan>
      <FilesExpire>0</FilesExpire>
      <MinFreeSpace>0.00000</MinFreeSpace>
   </Storage>
</Upkeep>
</Settings>
```

encres <num_entrada>

Descripción Solicita las resoluciones disponibles de una entrada. La respuesta consiste de una cadena de caracteres formateada en XML con las resoluciones de video a elegir, correspondiendo a la siguiente DTD:

```
<!--EncodingRes>
<!ELEMENT EncodingRes (Res?)>
<!ELEMENT Res (#PCDATA)>
<!ATTLIST Res
mode    (“NTSC” | “PAL”) REQUIRED
name    #CDATA           REQUIRED>
```

Parámetros *<num_entrada>*

Ejemplo **>encres 0** *(Solicita las posibles resoluciones de la primera entrada)*

```
<EncodingRes>
   <Res mode="NTSC" name="352x240">23068912</Res>
   <Res mode="NTSC" name="320x240">20971760</Res>
   <Res mode="NTSC" name="176x112">11534448</Res>
   <Res mode="NTSC" name="160x112">10485872</Res>
   <Res mode="PAL" name="384x288">25166112</Res>
   <Res mode="PAL" name="352x288">23068960</Res>
   <Res mode="PAL" name="176x144">11534480</Res>
   <Res mode="PAL" name="192x144">12583056</Res>
</EncodingRes>
```

---

curmode <num_entrada>

Descripción Consulta cual es el formato establecido para cierta entrada.
1: NTSC 2: PAL

Parámetros *<num_entrada>*

Ejemplo

```
>curmode 4
1
```

(El modo de captura está configurado como NTSC)

---

curres <num_entrada>

Descripción Solicita el código de la resolución de video establecida para la entrada.

Parámetros *<num_entrada>*

Ejemplo **>curres 1** *(Obtiene el código de la resolución de video de la segunda entrada)*

```
20971760
```

---

curresa <num_entrada>

Descripción Solicita la resolución de video establecida en una cadena de caracteres con el formato “NxM” donde N es el ancho en pixeles y M la altura en pixeles

Parámetros *<num_entrada>*

Ejemplo **>currresa 1** *(Solicita la resolución en la que captura la segunda entrada)*

```
320x240
```

---

curfile <num_entrada>

Descripción Consulta cual es el nombre completo del archivo en el que se encuentra grabando la entrada.

Parámetros <*num_entrada*>

Ejemplo
>curfile 2
D:\Captures\In3\In3_Channel3_20111003_101120.mp4

---

curstat <num_entrada>

Descripción Consulta el estado en el que se encuentra la entrada
0=OFF
1=INITIALIZED
2=IDLE
3=RECORDING
4=ERROR

Parámetros <*num_entrada*>

Ejemplo
>curstat 5
3 *(La sexta entrada está grabando)*

---

curname <num_entrada>

Descripción Solicita el nombre asignado a la entrada consultada

Parámetros <*num_entrada*>

Ejemplo
>curname 3
WRTV *(Es el nombre asignado a la cuarta entrada)*

---

interval <num_entrada>

Descripción Consulta la duración del segmento de la entrada, la respuesta es en milisegundos

Parámetros <*num_entrada*>

Ejemplo
>interval 2
1800000 *(Los segmentos de la tercer entrada duran 30 mins)*

Apéndice D

Configuración Optibase Player 400

La aplicación Optibase Player 400 está formada por filtros para DirectShow que permiten la visualización del streaming de audio y video que transmite el SDK de Optibase para observar las señales ubicadas en la entrada de la tarjeta grabadora.

El medio de transporte de las señales es mediante RTP que es un protocolo implementado sobre UDP. Para inicializar una sesión de visualización desde el lado del cliente (o receptor) es necesario obtener una descripción de la sesión.

Esta descripción de sesión se establece mediante un SDP que se describe en el RFC4566 de la IETF. En esta sesión de transmisión se establece la dirección I.P. y puerto origen y la codificación del flujo de datos, así como el nombre de la sesión entre muchos otros descriptores.

Para que DirectShow sea capaz de interpretar un archivo SDP, es necesario instalar el Optibase Player 400 que está incluido en el SDK de Optibase. Una vez instalado es necesario configurarlo correctamente de acuerdo al software que desarrollé para que este funcione correctamente.

Para configurar esta aplicación, primero se debe acceder al diálogo de configuración mediante:

Inicio > Programas > Optibase Player 400> Configuration

Una vez abierto el diálogo se debe de configurar con los siguientes valores:

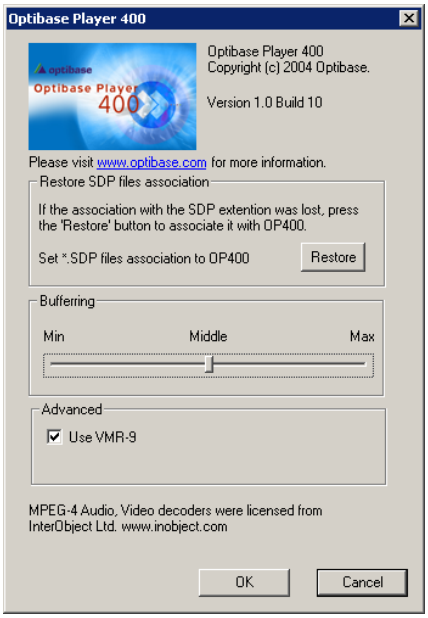


Figura D.1 - Configuración del Optibase Player 400

Al momento de dar clic en el botón de *Restore* la aplicación registra dentro del sistema operativo que los filtros proporcionados por Optibase son los más adecuados para administrar un archivo SDP.

El valor de *Buffering* consideré que el más apropiado era el de medio (*Middle)*, ya que en el valor mínimo (*Min),* el video se mostraba pasmoso con muchas pérdidas de cuadros en la visualización y en el valor máximo (*Max),* la señal que se observaba en el visualizador estaba desfasada por tres segundos, contra el desfase de un segundo en el valor medio.

A su vez, contemplé como una buena opción el utilizar la tecnología VMR-9 para el despliegue del video ya que esto agiliza un poco la forma en la que DirectShow reproduce el video.

Glosario

Audio Embebido Término que se refiere a un archivo multimedia que incluye tanto los datos de video como los datos de la señal de audio.

Codec Abreviatura de codificador-decodificador. En este caso se refiere a software utilizado para convertir un flujo de datos de las señales de audio y video en archivo y viceversa.

Disc Video Recorder Dispositivos de captura de video que graban en DVD's o en un disco duro local.

GraphEdit Programa incluido en la instalación de DirectShow y permite la creación y uso de gráficas de una manera visual.

GUI *Graphic User Interface*, del inglés, Interfaz Gráfica de Usuario.

IETF *Internet Engineering Task Force* es una organización internacional dedicada a la normalización de la Internet y ofrecen documentos que influencian en la forma en la que la gente debe de diseñar, utilizar y administrar la Internet.

MES *Microsoft Exporting Services* es el SDK utilizado para crear videocips durante un proceso de exportación en el sistema.

NAS *Network Attached Storage* es una tecnología de almacenamiento masivo que comparte la unidad de almacenamiento de un servidor con diversos clientes a través de una red.

NTFS *NT File System* es el sistema de archivos que utiliza Windows desde su versión 2000.

RTP *Real-time Transfer Protocol* es un protocolo de comunicación utilizado para la transmisión de audio y video en tiempo real.

SAN *Storage Area Network* es una red que conecta por medio de fibra óptica servidor y arreglos de discos con el propósito de compartir los recursos entre los elementos que la conforman.

Señal de Retorno Señal audiovisual que recibe el teleauditorio.

SDK *Software Development Kit* es un conjunto de herramientas que le permite al programador desarrollar aplicaciones para un software o hardware en específico.

Streaming F*lujo de datos* se refiere a la distribución de audio y/o video por medio de una red de ordenadores, donde se pueden comenzar a reproducir las señales mientras estas se están transmitiendo al ordenador que las reproduce.

Token Es un componente léxico se refiere a las palabras contenidas en los comandos, donde su delimitador es un espacio en blanco.

Video Anamórfico Video capturado en una relación distinta a la de la fuente original, con lo que se deforma el video en su captura y puede volver a ser deformado durante su reproducción a su relación original.

Referencias

1. Hunt, Craig; **TCP/IP Network Administration**, 3a ed.;O'Reilly Media, 2002, E.U.A., cap. 1
2. Weise, Marcus, et al**; How Video Works**; 2a ed.; Focal Press; 2007, E.U.A., cap 6, pág. 57-70
3. Cormen, Thomas H., et al; **Introduction to Algorithms**, 2a ed; MIT Press, 2001, E.U.A. cap 12
4. Ray, Erik T., **Learning XML**, 2a ed.;O'Reilly Media, 2003, E.U.A, págs. 113-119
5. Rajkumar Buyya , **High Performance Cluster Computing: Programming and Applications,** Prentice Hall PTR, 1999, E.U.A., pág 4-15 (Modelos)
6. Rajkumar Buyya , **High Performance Cluster Computing: Programming and Applications**, Prentice Hall PTR, 1999, E.U.A., pág 17-23 (Paradigmas)
7. Pilgrim, Mark, **HTML5 *Up and Running*,** 1a ed;O'Reilly Media, 2010, E.U.A., cap 5.
8. **User Datagram Protocol, IETF Documents**
   http://tools.ietf.org/html/rfc768 (15/Mayo/2011)
9. **Interprocess Communications**, **MSDN**
   http://msdn.microsoft.com/en-us/library/aa365574(VS.85).aspx (24/Mayo/2011)
10. **Multi-threading with C++ and MFC, MSDN**
    http://msdn.microsoft.com/en-us/library/975t8ks0%28v=VS.80%29.aspx (30/Mayo/2011)
11. **Multithreading: How to Use the Synchronization Classes, MSDN**
    http://msdn.microsoft.com/en-us/library/thdxkfx9%28v=VS.80%29.aspx (30/Mayo/2011)
12. **Windows Programming/Handles and Data Type Wikibooks**
    http://en.wikibooks.org/wiki/Windows_Programming/Handles_and_Data_Types#HANDLE (27/sept/2011)
13. **OpenProcess function, MSDN**
    http://msdn.microsoft.com/en-us/library/ms684320(v=VS.85).aspx (26/Sept/2011)
14. **GetGuiResources function, MSDN**
    http://msdn.microsoft.com/en-us/library/ms683192(VS.85).aspx (26/Sept/2011)
15. **GetProcessHandleCount function, MSDN**
    http://msdn.microsoft.com/en-us/library/ms683214(VS.85).aspx (26/Sept/2011)
16. **File Systems, Microsoft Technet**
    http://technet.microsoft.com/en-us/library/cc938437.aspx
17. **Naming Files, Paths, and Namespaces, MSDN**
    http://msdn.microsoft.com/en-us/library/windows/desktop/aa365247(v=vs.85).aspx
18. **RFC4566 – SDP: Session Description Protocol**
    http://tools.ietf.org/html/rfc4566