UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE INGENIERÍA

CLIENTE DE VIDEOVIGILANCIA VIDEO WALL

TRABAJO PROFESIONAL

QUE PARA OBTENER EL TÍTULO DE

INGENIERO EN COMPUTACIÓN

PRESENTA:

SÁNCHEZ MALFAVÓN JUAN GERARDO

ASESOR

ING. ALEJANDRO VELÁZQUEZ MENA

CIUDAD UNIVERSITARIA

Indice

Índice de imágenes

Introducción

Cada vez se vuelve más indispensable que los gobiernos sean capaces de reaccionar a las actividades que ocurren durante el transcurso de un día y que sean capaces también de poder actuar de la forma más rápida y apropiada posible. También es muy importante para las empresas optimizar sus tiempos de entrega de productos, el brindar un servicio rápido es un factor para que los clientes sean fieles y también le crea a las empresas una buena fama, con la que seguro se hacen de más clientes. Es decir vivimos en un mundo en donde lo más importante es la optimización de cualquier tarea o actividad que se realice y por ende se tienen que buscar técnicas que faciliten la optimización de dichas actividades.

Uno de los sistemas en los que se pueden apoyar tanto empresas como gobierno, es en un sistema de video vigilancia, ya sea que en alguna vialidad importante haya ocurrido algún accidente, que se estén llevando a cabo manifestaciones o que esté ocurriendo algún asalto, con un sistema de este tipo se puede planear como actuar dependiendo de la situación y de su gravedad. También en un momento dado, se pueden llegar a generar estadísticas para que en lugar de que se espere a que ocurra algo, se pueda actuar anticipadamente contra ese algo. O en el caso de las empresas, no se puede estar cuidando a todos los empleados que laboran en ella y tener a una persona que verifique los tiempos de operación de cada una, eso generaría un gasto enorme y en algunos casos sería prácticamente imposible de realizar.

Por ejemplo si una empresa se dedicara a repartir productos en diferentes puntos del país y si en determinado momento el dueño de la empresa se diera cuenta de que los tiempos de entrega varían mucho aun cuando se hagan a los mismos puntos todas las semanas, con un sistema de video vigilancia y cámaras con tecnología **3G** se podrían saber las razones de esos desfases en cuanto a tiempos de entrega.

En el mercado actual existen muchos sistemas de vídeo vigilancia que permiten realizar monitoreo y la descripción del sistema que a continuación voy a presentar, es uno de ellos, pero este tiene ciertas diferencias con los otros sistemas que se ofrecen hoy en día, la diferencia principal es que brinda todo un ecosistema en el que se puede lograr dar solución a situaciones como las antes mencionadas, un ecosistema que le permite brindar una mejor experiencia ya que en el conviven todos los módulos que forman parte de un sistema de video vigilancia, desde las cámaras, hasta el cliente y el propio servidor.

Además de esta convivencia, la interfaz de usuario tiene el plus de ser muy intuitiva, a tal grado que cualquier persona podría ser capaz de utilizarlo de forma correcta, ya que solo el administrador, que se considera que cuenta con los conocimientos necesarios y en caso de no ser así se le capacita, es el único encargado de hacer el levantamiento de las cámaras y de crear los perfiles de usuario, quien manipule el cliente solo tendrá que agregar las cámaras sin necesidad de proporcionar información alguna para que ésta comience a visualizar video.

Este escrito está conformado por cuatro capítulos principales, el primer capítulo está dedicado a la empresa en la que laboro, en este capítulo comenzaré hablando sobre el marco referencial de la empresa, mencionando un poco de su historia, la cantidad de empleados que en ella laboran y los productos y soluciones que la empresa brinda, posteriormente describiré la forma en la que está estructurada la empresa y del área en la que me desempeño, para por último hablar un poco sobre los clientes con los que tiene relación Global Corporation.

En el segundo capítulo, haré un resumen de los proyectos en los que he participado y de cuáles son las actividades que he llevado a cabo en dichos proyectos. El capítulo tres es el más importante de todos ya que en el hago la descripción del proyecto que utilicé como tema principal de este escrito. Este capítulo lo comenzaré, hablando sobre los elementos principales que permiten que este software funcione, posteriormente pasaré a hablar de las tres principales versiones que se han desarrollado de este, mencionando las capacidades y limitaciones de cada una, así como los problemas y soluciones que han surgido a lo largo de su desarrollo. Una vez haya terminado de hablar sobre la última versión, descompondré esta en los módulos que la conforman para también describir cada uno de estos. Este capítulo finalizará con la descripción del *back-end* que respalda al *front-end* del *software*.

Para finalizar, en el cuarto capítulo haré una descripción de los resultados que se han obtenido hasta el momento así como también describiré las partes que se encuentran en desarrollo y por ultimo daré mis conclusiones acerca del estado actual del *software* y de la evolución que tendrá en un futuro.

Capítulo 1. Organigrama

Global Corporation International S.A. de C.V. es un grupo de empresas que ofrecen soluciones integrales con nuevas tecnologías. Las soluciones de *software* que presentan y su implementación están planeadas para su uso en los sectores más estratégicos del gobierno, ofreciendo estas soluciones para su utilización en la seguridad pública, aunque también cuenta con soluciones ofrecidas a empresas y a hogares, con lo cual tiene la capacidad de cubrir todo el mercado en cuanto a seguridad se refiere. Global Corporation tiene un acuerdo con Grupo Carso para ofrecer la seguridad como un módulo más de los servicios que ofrece dicho grupo.

Global Corporation International S.A. de C.V. nació en el año 2004 en el estado de Chihuahua, posteriormente, en el año 2008 abrieron oficinas en el Distrito Federal, en cuyo momento la empresa paso a ubicarse en la Delegación Miguel Hidalgo, en Sierra Gorda No. 12, en este sitio comenzaron a desarrollarse los proyectos que más éxito han tenido en la empresa y que son precisamente los que están encaminados hacia la seguridad.

Hoy en día las oficinas de Global Corporation se encuentran ubicadas en Lago Zúrich No. 245, edificio Falcón piso 19 (Imagen 1.0), de igual manera en la delegación Miguel Hidalgo. Una de las empresas que forma parte de Global Corporación es Global Human Services, de la cual yo formo parte desde el 27 de Febrero del 2012, dicha empresa está conformada por un total de 104 personas y su giro es el desarrollo de *software* y *hardware*.

Imagen 1.0 Lago Zúrich No. 245

Global Corporación cuenta con una gran cantidad de productos y servicios, entre los que destacan:

- Video vigilancia
- Reconocimiento facial
- Conteo de personas
- Control vehicular

Lo mejor de estos proyectos es que la mayoría de ellos son modulares, por lo que el cliente le puede indicar a la empresa que es lo que necesita y entonces se unen los módulos que sean necesarios para poder satisfacer sus peticiones. En cuanto a la estructura organizacional Global Corporation ésta formada por nueve áreas y cuenta con un director general (Imagen 1.1).

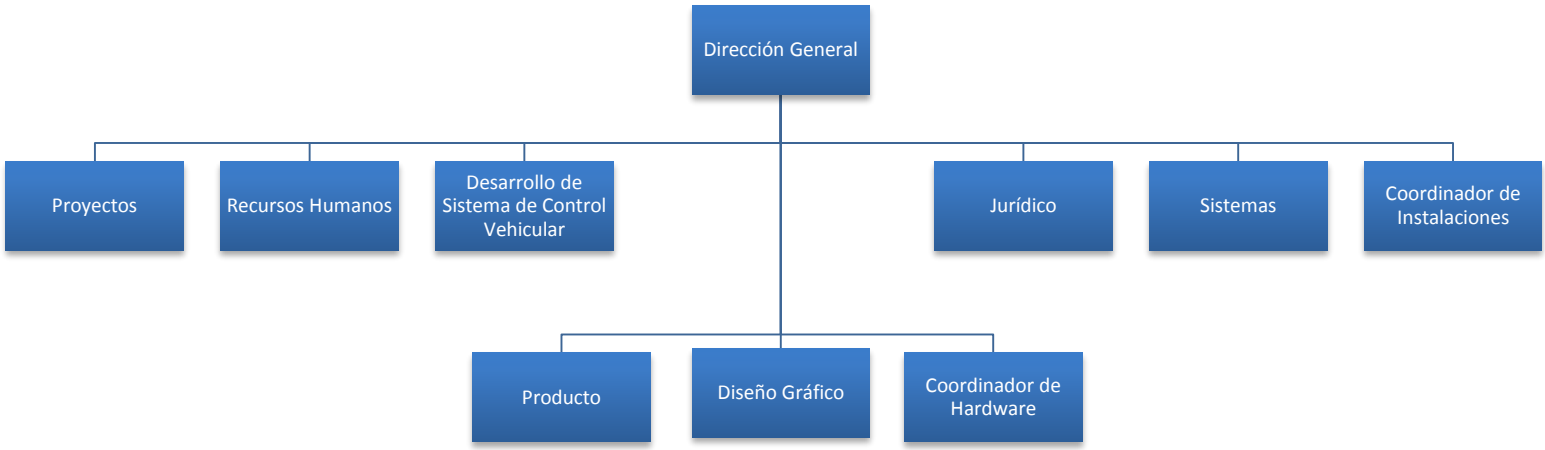


Imagen 1.1 Organigrama general

El área de Proyectos esta subdividida en otras áreas como la de becarios tanto de *iOS* como de desarrollador **.NET** y desarrolladores de *software* junior entre otras, Recursos Humanos obviamente se encarga de la contratación del personal, Desarrollo de Sistemas de Control Vehicular desarrolla un *software* que es capaz de leer las placas de un vehículo y a partir de ahí genera multas dependiendo de la falta que ha cometido el conductor, Producto se encarga del *marketing* de los productos y servicios que la empresa ofrece, Diseño Gráfico realiza las presentaciones que el Director General se encarga de mostrar a los clientes, Coordinador de Hardware diseñan y construyen los diferentes dispositivos que la empresa vende o utiliza internamente, Jurídico obviamente se encarga de las cuestiones legales de la empresa, Sistemas se encarga de la administración de los sistemas operativos y del soporte a los usuarios de las computadoras, Coordinación de Instalaciones planea y lleva a cabo la instalación de los productos como son las cámaras de vídeo vigilancia y el *software* utilizado para realizar dicha vídeo vigilancia.

Yo pertenezco e inicie mi trabajo en el área de Proyectos, más específicamente en el área de becarios .NET, mi jefe directo es el responsable de toda el área de Proyectos, en el área de becarios es en donde se está desarrollando el *software* que en este escrito estoy presentando, además de otros programas tales como reconocimiento facial y contador de personas. El *software* que presento aquí, lo he desarrollado prácticamente desde el inicio del mismo, para la tercera versión yo me encargué de proponer su estructura y los diferentes módulos en los que se divide.

Ahora bien antes de hablar de los proyectos en los que he participado, en cuanto a clientes se refiere, Global Corporación también tiene varios, entre los que destacan Brasil, y los gobiernos de Tijuana, Saltillo y Puebla, ya que en estos es en donde mi *software* va a ser mayormente utilizado. Pero también cuenta con clientes como Sears, Sanborns o **Praxair**, en los que mi proyecto toma un enfoque diferente, ya que estos buscan utilizar este *software* como un generador de estadísticas más que como lo que es, un sistema de video vigilancia. Y es aquí donde se ve la modularidad de los proyectos ya que mi *software* se combina con el contador de personas para poder obtener estas estadísticas y así presentarlas al usuario. Para el caso de Praxair y gracias a que los Drones que ofrece Global Corporation tienen capacidades para soportar redes 3G, lo que se busca es verificar los tiempos de entrega de los productos que ofrece y conocer las razones por las cuales estos tiempos de entrega varían tanto.

Capítulo 2. Descripción de Proyectos

Desde mi contratación en Global Human Services hasta el día de hoy, he participado en un total de tres proyectos, los cuales han sido bastante interesantes, ya que por el enfoque que tienen han demandado de mi como programador el preocuparme por todos y cada uno de los aspectos que pueden afectar el rendimiento tanto del sistema operativo en el que es instalado como en el del *software* que he desarrollado. Esto también implica que los conocimientos que he adquirido son muy elevados, permitiéndome hasta el momento, poder satisfacer los objetivos que se me han impuesto en la empresa.

Los proyectos en los que he participado son: una cámara de video llamada *Drone*, un contador de personas y por último un cliente de video llamado *Drone VideoWall*. A continuación haré la descripción de cada uno de estos proyectos.

Proyecto Drone

Este fue el primer proyecto en el que participé y pienso que fue uno de los proyectos más difíciles que tuve, mi objetivo en este proyecto fue el de la creación de una cámara de video “inteligente”. Para darle vida a estas cámaras, la empresa adquirió unas tarjetas de desarrollo llamadas Beagleboard (Imagen 2.0), las cuales son producidas por **Texas Instruments** en asociación con **Digi-Key and Newark element14** y que tienen como especificaciones principales las siguientes:

- 512 MB de memoria RAM
- Procesador ARM Cortex-A8 a 1 GHz
- 4 puertos USB
- Puerto Ethernet
- Puerto microSD
- Soporte para cámara
- Jacks estéreo de entrada y salida

Imagen 2.0. Tarjeta de desarrollo Beagleboard

Las especificaciones mencionadas, hacen notar que se trataba de una tarjeta de desarrollo relativamente limitada para las cuestiones a las que iba encaminada, por ende se necesitaba un sistema operativo que pudiera ofrecer un buen rendimiento y que al mismo tiempo pudiera lograr lo que se quería, darle inteligencia a dichas tarjetas. La empresa eligió un sistema operativo llamado *Amstrong*, el cual es una distribución de Linux basada en Debian y la razón de su elección fue debido a que es un sistema operativo ligero, además de que la página de la tarjeta de desarrollo lo recomendaba.

El objetivo final de este proyecto era el de poder hacer que dichas cámaras pudieran hacer procesamiento de imágenes, para utilizarlas en proyectos relacionadas con analíticos de video, como son el reconocimiento facial y el conteo de personas. Pero el objetivo a corto plazo que se me asigno fue el de poder visualizar o mejor dicho el de hacer que la tarjeta de desarrollo se comportara como una cámara de video. Para lo cual lo primero que hice fue la instalación de dicho sistema operativo, esta instalación la lleve a cabo en una tarjeta *micro-SD* ya que estas tarjetas de desarrollo no cuentan con un disco duro. Una vez instalado el sistema operativo, empecé a hacer levantamiento de servicios tales como *SSH*, *apache*, *MySQL* entre otros.

Para lograr la instalación de este sistema en una *micro-SD*, se creaban dos particiones diferentes que en el momento de encender la tarjeta de desarrollo, una de las particiones correspondía con el sistema de archivos que utiliza **Linux**, en esta partición almacené algunos *scripts* para carga de valores por *default* que necesitaba para la configuración de la cámara, la otra partición almacenaba archivos de configuración propios del sistema. La cámara que se utilizó en ese momento fue la cámara LI-5M03 (Imagen 2.1), misma que tenía una resolución de 5MP y fue construida por **Leopard Imaging Inc.**

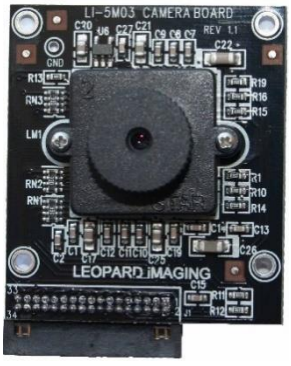


Imagen 2.1. Cámara LI-5M03

La primera vez que instalé el sistema operativo, no reconoció la cámara, aunque dicho sistema se suponía que ya estaba adaptado para trabajar con esa cámara, en su momento pensé que se trataba de un problema de *drivers* y empecé a investigar y a probar soluciones que iba encontrando durante mis investigaciones, pero nada funcionó. Después de estos intentos, entre a un canal donde se formaban grupos de investigación y desarrollo y contacte con un ingeniero de otro país mismo que me oriento basándose en lo que él había realizado, al final el problema que causaba que la cámara no fuera reconocida era que en la partición del sistema, se tenía que agregar una directiva que le indicaba al sistema que quería hacer uso de la cámara, una vez hecho esto, tuve que recompilar el **Kernel**. Cuando terminé de recompilarlo, la cámara ya aparecía en el directorio *dev/* del sistema de archivos, por lo que procedí a hacer uso de **gstreamer**, para visualizar el video que estaba captando la cámara.

Una vez que logré que el video se mostrará, se me pidió realizar una interfaz gráfica para que hubiera una forma visual de configurar la cámara "inteligente", esta interfaz se mostraría en una página *web* en la cual podría ingresar toda aquella persona que tuviera los permisos necesarios para realizar cambios en la configuración de dicha cámara.

Los parámetros que se podían configurar fueron principalmente la resolución de la cámara, los **fps** a los que transmitía la cámara que estuviera siendo utilizada, la compresión y en un futuro iba a permitir mover la cámara o también programar un patrón que la cámara debería seguir en un tiempo establecido, esto gracias a que se estaba trabajando en una serie de motores para proporcionarle a las cámaras las capacidades de ser cámara ***PTZ***.

Una vez que termine de realizar la interfaz gráfica que se mi pidió, fui cambiado de área, al área de video, desarrollando para la plataforma de **.**NET, en la que comencé a desarrollar una aplicación de conteo de personas.

Contador de personas

Cuando se me asigno este proyecto, lo primero a lo que me dediqué fue a realizar un investigación de que ***APIs*** o ***SDKs*** había disponibles para poder cumplir con el propósito de contar a las personas que iban pasando por el área de visualización de las cámaras de Global Corpotation. Lo primero que encontré fue ***OpenCV*** pero el inconveniente que tuve con *OpenCV* fue el hecho de que solo admitía los lenguajes de programación *C++*, *Phyton* y *Java*, pero yo estaba desarrollando el software de video vigilancia en el lenguaje *C#* por lo que esta librería en si no me funcionó. Pero mientras investigaba di con un *wrapper* (envoltorio) de la librería de *OpenCV* (Imagen 2.2) llamada ***EmguCV*** (Imagen 2.3), el cual si era compatible con lenguajes de .NET más específicamente con el lenguaje en el que yo estaba desarrollando.

Imagen 2.2. Open CV

Imagen 2.3. Emgu CV

También encontré un ***Framework*** llamado ***Aforge.NET*** (Imagen 2.4) el cual era completamente orientado al *Framework* de .NET. Una vez encontrada la información que necesitaba empecé a comparar y ver qué era lo que más me convenía para poder desarrollar el programa con los requerimientos que se me pedían. Al final me pareció más conveniente desarrollar en *Emgu CV*.

La primera versión que desarrollé fue un desastre, ya que no contaba con los conocimientos necesarios sobre procesamiento de imágenes. Considero que en este tema aún no tengo dichos conocimiento, ya que solo utilicé un poco de lógica para poder realizar un conteo de persona más o menos decente mismo que platicaré más adelante, pero con los nuevos compañeros que entraron al área después de que a mí me cambiaron de proyecto me enseñaron que se podían hacer muchas más cosas con estas librerías.

Imagen 2.4. Aforge.NET

Cuando una persona estaba en el área de visión del contador de personas en su primera versión, éste podía ser capaz de contarla un millón de veces si la persona permanecía en dicha área de visión, por lo que esta primera versión no distinguía si la persona que veía en un primer *frame* era la misma en tres *frames* posteriores. Y como tal este *software* nunca reconoce si es la misma persona la que permanece en la escena, pero para dar una solución a este problema lo único que se me ocurrió en ese momento, fue el tomar una distancia como radio para comparar entre el cuadro que enmarcaba a una persona cuando estaba en el área de visión y compararla con la enmarcada en el siguiente *frame* (Imagen 2.5) y si la distancia establecida no había sido superada, entonces se consideraba como la misma persona y ya no era contada.

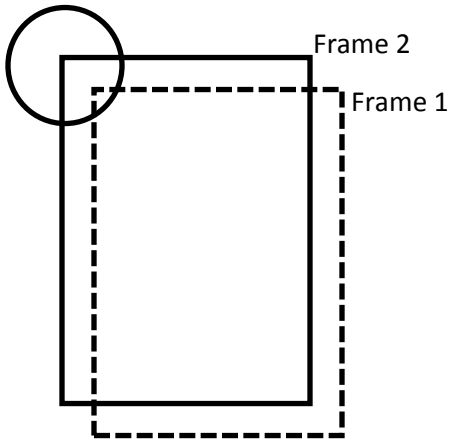


Imagen 2.5 Comparación de *frames*

Después de aplicar esta solución, el conteo de personas mejoró pero aún tenía fallos ya que el sistema se confundía con la dirección que llevaban las personas y también se confundía cuando las personas que pasaban por el área de visión de la cámara eran demasiadas. Más tarde logré implementarle a este proyecto la capacidad de saber si la persona que estaba en el área de visión estaba entrando o saliendo de algún lugar o mejor dicho, le di la capacidad de saber la dirección de la persona que iba pasando por su área de visión y a partir de ahí determinar si estaba entrando o saliendo de una área determinada.

Esta capacidad de saber la dirección la implemente mediante el dibujo de dos líneas de las cuales una de ellas se establecía como origen y otra como destino (Imagen 2.6), así si el origen estaba establecido en la entrada de algún establecimiento, y el destino estaba fuera del establecimiento, solamente tenía que hacer una comparación entre estos dos puntos y si la primer línea que se tocaba era la de destino quería decir que la persona estaba entrando, por el contrario si la primer línea que se tocaba era la del origen, quería decir que la persona estaba saliendo.

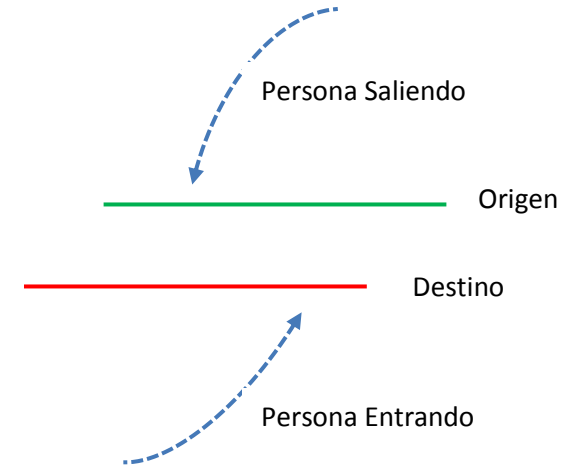


Imagen 2.6 Detección de entrada-salida

El aspecto visual del contador de personas era el mostrado en la Imagen 2.7, en el cual como se puede apreciar era demasiado sencillo, pero en realidad el objetivo para mí no era el hacer que se viera agradable al usuario, sino que fuera capaz de realizar un conteo decente de personas. En realidad ni siquiera se necesitaba de una interfaz gráfica como tal, ya que cada persona contada era enviada a una base de datos, y un cliente *web* que visualizaba la misma cámara era el que mostraba los resultados del conteo.

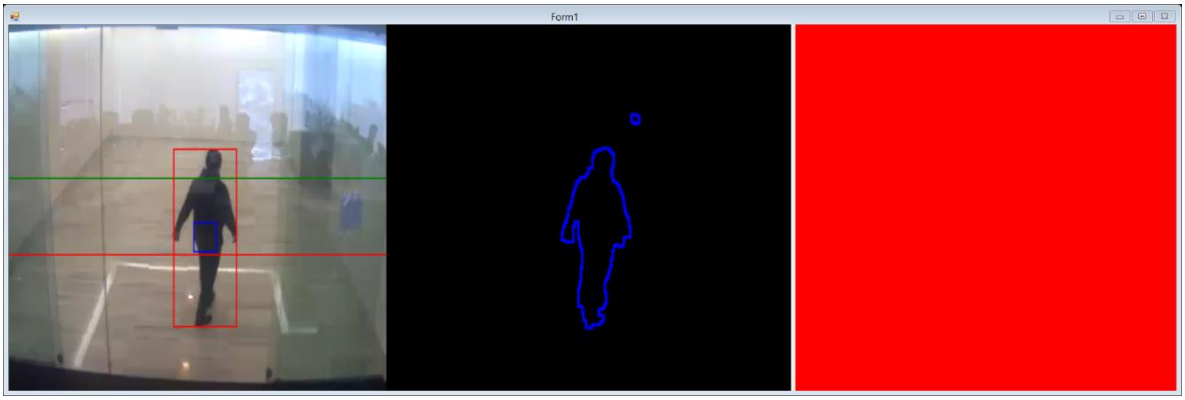


Imagen 2.7 Contador de personas

Este *software* fue encaminado principalmente a las cadenas comerciales, más específicamente, para tiendas como Sears y Sanborns, aunque hasta el momento también ha sido utilizado en soluciones para el hogar, pero en este caso toma un enfoque diferente, un enfoque en el que el *software* es tomado como un sistema de alertas. Para esto hice que el *software* conviviera con otros sistemas y plataformas como son las móviles, en éste caso en el momento en el que había un movimiento o mejor dicho, en el momento en el que la cámara detectaba un movimiento en su campo de visión se comunicaba con un servicio *web*, mismo que se encargaba de enviar notificaciones a los dispositivos móviles soportados (Android y iOS) para que el usuario pudiera tomar la decisión que más creyera conveniente, dependiendo de lo que estaba ocurriendo en el lugar en donde ocurrió el evento, ya que el dispositivo móvil era capaz de visualizar el video que estaba siendo transmitido por las cámaras que el cliente contrató para su propiedad.

Ahora bien, hablando un poco más sobre las cadenas comerciales, el propósito de instalar cámaras en esos lugares para que realizarán dicho conteo, fue para generar estadísticas de la cantidad de personas que entran a cierta área de la tienda, así como también la cantidad total de personas que entran y salen de la misma para que pudieran utilizar estrategias de mercado para poder vender aún más. Estas estrategias de mercado, se podrían ver reforzadas con otro de los módulos que la empresa estaba desarrollando y que posteriormente sería incluido como parte del entorno de aplicaciones dedicadas al análisis de video, este módulo era el de reconocimiento facial (Imagen 2.8) con el cual se podría reconocer al cliente y dependiendo de sus consumos poder ofrecerle noticias, promociones o descuentos para así conseguir que sus ventas incrementen, estas ofertas o promociones serían enviadas a su *Smartphone* para tener un contacto más rápido y directo con el cliente.

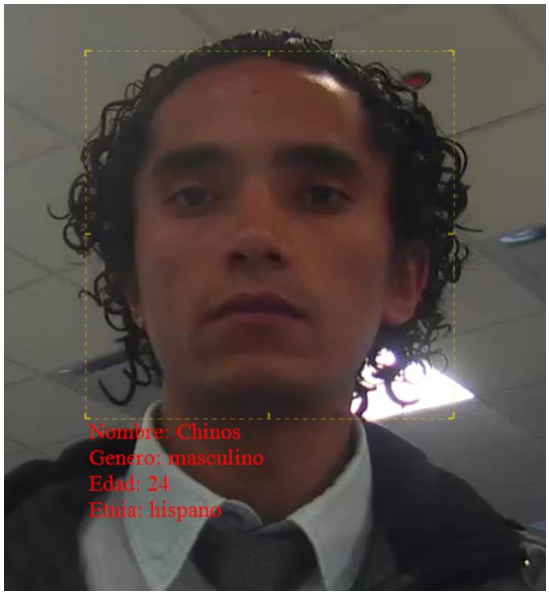


Imagen 2.8 Reconocimiento facial

En caso de que el *software* no sea capaz se saber de qué persona se trata cuando intenta hacer el reconocimiento facial, hecho que pasará cuando se inicie el *software* por primera vez, y se mantendrá así hasta que el cliente acepte proporcionar sus datos, el comportamiento de este *software* será el de realizar una estimación (Imagen 2.9) de la edad, género y etnia de la persona que esté en el área de visión de la cámara y que haya detectado su rostro.

Imagen 2.9 Estimación de características de persona

Una vez que programé la solución del radio de distancia para comprobar si se trataba o no de la misma persona, fui cambiado nuevamente de proyecto, aun desarrollando en el *Framework* de .NET bajo el lenguaje *C#*. Pero esta vez se trató de un cliente video vigilancia llamado *Drone VideoWall*.

Drone VideoWall

Imagen 2.10 Drone VideoWall

La Imagen 2.10 que se muestra arriba, es la interfaz gráfica de la última versión de este proyecto y ya que decidí elegir éste como proyecto principal debido a que es el *software* en el que he invertido mayor cantidad de tiempo de desarrollo, llegando a construirlo prácticamente desde cero, analizando los módulos en los que podía ser dividido y aplicando todos los conocimientos que tenía y también todos aquellos que fui obteniendo durante el desarrollo del mismo, solo mencionaré algunas de las capacidades que tiene el *Drone VideoWall* y en el siguiente capítulo desglosaré cada una de estas capacidades y de todas aquellas que no mencioné.

Este *software* es un sistema de video vigilancia que hace uso de la arquitectura cliente servidor, estos dos componentes trabajan de forma dedicada, es decir, que todo está adaptado para que el rendimiento del cliente y el servidor sea muy bueno tomando en cuenta la cantidad de recursos demandados al momento de realizar transmisión y visualización de video en vivo.

Sin embargo y por motivos de demostración, también programé un módulo en el que se pueden agregar cámaras por conexión directa a las mismas, aunque no es lo más recomendable ya que algunas de las capacidades que tiene en conjunto el cliente y el servidor, podrían verse afectadas limitando la experiencia de usuario y también el consumo de recursos del *software*, por mencionar algunos ejemplos el movimiento *PTZ* no podría realizarse directamente sobre la cámara así como también la programación de rutinas de dicho movimiento y el almacenamiento de vídeo entre otros.

El sistema es capaz de visualizar un total de 20 cámaras de forma simultánea, aunque lo recomendable son alrededor de nueve cámaras ya que el ser humano no es capaz de poner atención a tantas cámaras a la vez. Por este motivo se realizó la implementación de vistas, ya que por medio de estas podemos ver menor cantidad de cámaras con un mayor tamaño lo que facilita el monitoreo. Aunque las capacidades del *Drone VideoWall* sobrepasan este número, ya que cuenta con un sistema de organización en pestañas y en cada una de estas se pueden ir agregando más cámaras. Bajo pruebas de estrés, este *software* fue capaz de soportar un poco más de sesenta cámaras, aunque se sigue trabajando para poder lograr que el sistema pueda soportar una mayor cantidad.

Capítulo 3. Proyecto Principal

Drone VideoWall

Objetivos del proyecto

El sistema permitirá simultáneamente:

- Mostrar video en vivo
- Controlar cámaras *PTZ*
- Grabar video
- Reproducción de video grabado
- Detección de movimiento
- Control de usuarios
- Niveles de usuarios

Descripción del proyecto

Inicie mi participación en éste proyecto alrededor de seis o siete meses después de que entré a esta empresa, en ese momento el proyecto *Drone VideoWall* estaba muy limitado, apenas se estaba empezando a desarrollar y se presentaron muchísimos problemas relacionados con el rendimiento del *software*. En la última versión que se ha desarrollado de este sistema, el rendimiento a mejorado enormidades y en general todo el sistema ésta mucho mejor, desde la interfaz hasta la robustez y desempeño del mismo.

Los elementos principales que hacen posible el funcionamiento y uso de este software son los siguientes:

- *Framework* propietario de *Streamcoders*.
- Códec de video **H.264**.
- Protocolo **RTSP**.
- *Drone VideoWall* (*back-end*, *front-end*).

Streamcoders (Imagen 3.0) es la empresa a la que Global Corporation contacto para poder hacerse de un *Framework* llamado *MediaSuite.NET*, con el cual se lograron capacidades en el sistema de video vigilancia tales como la transmisión de video haciendo uso del códec H.264 y la grabación de dicho video para su posterior visualización. Este *Framework* es el núcleo del programa en cuanto al *code-behind* se refiere. Pero el hecho de utilizar este *Framework* no solo me ayudo a llevar a cabo el proyecto, sino que también implicó la aparición de ciertas dificultades, ya que la empresa *Streamcoders* no proporcionaba una buena documentación, esto con el objetivo de poder obtener más ingresos por el soporte que yo como programador necesitaba, pero que debido a los costos de este soporte, no pude obtener dicha capacitación y tuve que enfrentarme a estos problemas por mi propia cuenta.

Pero a pesar de eso se decidió optar por este *Framework* debido a que cuenta con gran prestigio, ya que brinda sus servicios a empresas tales como AT&T, IBM, CiscoSystems, Microsoft, LG, entre otros.

Imagen 3.0 Clientes Streamcoders

La utilización del códec de video H.264 trajo grandes beneficios, ya que éste parte de lo que hace y lo que más nos benefició es que, por poner un ejemplo, cuando se ésta haciendo la transmisión de video, si la cámara no tiene un cambio en cuanto a lo que ésta visualizando, el *frame* no se actualiza y si llega a ocurrir un cambio el códec es capaz de saber que parte del *frame* es la que se ésta moviendo y solo actualiza dicha parte. Con esto se logró un gran ahorro en la transmisión y debido a que los Drones que son utilizados con ciertos clientes, tienen la capacidad de realizar la transmisión de video mediante la red 3G, ésta característica del códec también logró un gran ahorro en la parte económica.

El protocolo *RTSP* me proporciono las facilidades para mantener las sesiones de video que se establecían entre el cliente y el servidor activos controlando la sincronización de los flujos de datos (video en este caso).

Antes de hablar del proyecto en su última versión estable, primero haré una descripción de cómo ha cambiado con el tiempo este proyecto, es decir hablaré de las diferentes versiones que se han construido mencionando las capacidades con las que contaban, así como también los problemas que durante su construcción fueron presentándose y como estos fueron resueltos. Cabe mencionar también que esta última versión ésta compuesta por dos diferentes **soluciones**, una se encarga de dar soporte a la parte gráfica mediante animaciones y agregado de controles, y la otra solución se encarga de dar soporte a todos los aspectos relacionados con el video y su correcta transmisión. El nombre de estas soluciones son *VideoClient* (*front-end*) y *VideoCoreLibraries* (*back-end*) respectivamente, y en la última versión que se describe en este escrito se presentarán y comentarán ambas soluciones.

Versiones

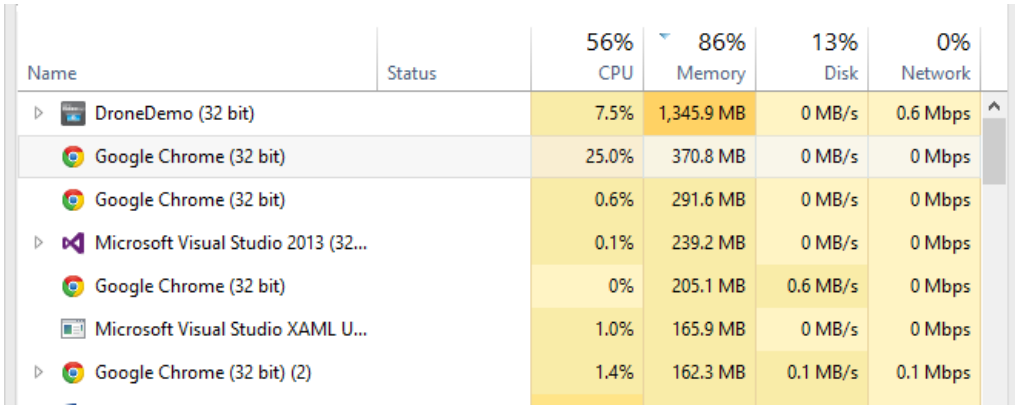
Primera versión

La primera vez que tuve contacto con este *software*, se estaba desarrollando su primera versión y se había elegido *Windows Forms* para desarrollar el proyecto, al ser la primera versión y por estar en sus inicios de desarrollo la interfaz gráfica (Imagen 3.1) que presentaba era muy pobre, los controles no estaban personalizados, prácticamente eran los mismos controles que aparecen en un proyecto creado por default en *Visual Studio*, el proyecto no contaba con un nombre oficial y ni siquiera contaba con un icono que lo identificará.

Imagen 3.1 Interfaz gráfica primera versión

Lo que era aún peor era la parte de las capacidades que este tenía. Las cámaras que se querían visualizar, eran agregadas mediante código duro, por lo tanto también el dinamismo era limitado ya que no se podían cambiar las cámaras, tampoco re-conectarse a ellas si la comunicación se perdía o eliminarlas en caso de que no la quisiéramos seguir viendo. Estos aspectos fueron los que yo empecé a desarrollar, todavía sin meterme con la interfaz de usuario del proyecto. También le agregue el *ComboBox* que aparece en la imagen de arriba, el cual podía conectarse a un servicio *web* con el cual podía obtener las cámaras y entonces se tenía la capacidad de seleccionar la cámara requerida. En los aspectos relacionados al rendimiento, tampoco se tenían grandes resultados, ya que la librería de *Streamcoders* utilizada para la codificación y decodificación de los *frame* de video era completamente desconocida para mí y además la documentación que *Streamcoders* proporcionaba estaba muy limitada. Para utilizar dichas librerías se habían basado en un cliente de ejemplo que venía junto con la instalación de dicha librería y yo continúe con esa tendencia.

Como consecuencia de esto y de algunos *bugs* con los que contaba el proyecto por parte de los programadores que habían participado antes que yo en el proyecto, éste *software* solo era capaz de soportar cuatro cámaras de manera simultánea, de hecho solo soportaba tres porque en cuanto la cuarta empezaba a funcionar el cliente se congelaba, mostrando en el administrador de tareas (Imagen 3.2) que este era el proceso que más estaba consumiendo recursos de memoria *RAM*.



Name	Status	56% CPU	86% Memory	13% Disk	0% Network
DroneDemo (32 bit)		7.5%	1,345.9 MB	0 MB/s	0.6 Mbps
Google Chrome (32 bit)		25.0%	370.8 MB	0 MB/s	0 Mbps
Google Chrome (32 bit)		0.6%	291.6 MB	0 MB/s	0 Mbps
Microsoft Visual Studio 2013 (32...		0.1%	239.2 MB	0 MB/s	0 Mbps
Google Chrome (32 bit)		0%	205.1 MB	0.6 MB/s	0 Mbps
Microsoft Visual Studio XAML U...		1.0%	165.9 MB	0 MB/s	0 Mbps
Google Chrome (32 bit) (2)		1.4%	162.3 MB	0.1 MB/s	0.1 Mbps

Imagen 3.2 Consumo de memoria

Como consecuencia no servía prácticamente de nada el hecho de conectarse al servicio web para obtener el listado de cámaras disponibles ya que no se podían aprovechar todas las cámaras. Al hablar de la calidad de video que se mostraba, éste también presentaba errores y estos comenzaban desde antes de iniciarse el programa, y es que al parecer los *frames* se iban almacenando en un *buffer* y en el momento en el que el programa se hacía visible al usuario, el *buffer* se vaciaba mostrando todos los *frames* que se habían almacenado en éste y los mostraba demasiado rápido durante unos segundos, hasta que se alcanzaba el *frame* actual es decir hasta que el *buffer* era vaciado y solo entonces el despliegue de *frames* era normal. Otro de los problemas que se presentaba era que había ocasiones en el que en el lienzo (en ese momento no tenía nombre pero posteriormente se llamó VideoDisplay) se mostraba el *frame* 1, después seguía con la visualización normal mostrando el *frame* número 2, pero en vez de continuar mostrando el *frame* 3 este regresaba al *frame* 1 y lo volvía a mostrar, después ya mostraba el *frame* 3 y después regresaba al dos y así continuaba todo el tiempo que el video era visualizado.

Otro aspecto del que me di cuenta fue de que cuando cerraba el programa por medio del botón de cerrado como lo hace cualquier otro *software* los recursos que eran ocupados por este programa no eran liberados como se esperaría que se hiciera y esto ocurría por que el proceso de este *software* seguía apareciendo en los procesos del administrador de tareas, lo que provocaba que los recursos que este *software* consumía no pudieran ser utilizados por otros programas a menos que la computadora en la cual estaba instalado el *software* fuera reiniciada. En ese entonces el *software* no era para nada bueno, ni siquiera se acercaba un mínimo a uno que pudiera ser lanzado como una versión beta del producto, incluso había ocasiones en el que el consumo de recursos del programa era tan grande que el sistema completo terminaba congelándose.

Segunda Versión

La segunda versión (Imagen 3.3) de este *software* tuvo un enorme cambio en comparación con la primera versión, pero en realidad en este punto se construyeron dos proyectos distintos de esta segunda versión, esto con el objetivo de probar cuál de las dos versiones se comportaba mejor y bajo qué situaciones pero al final, en la última versión presentada más adelante se optó por unir estos dos proyectos ya que uno le daba solución a los problemas del otro y viceversa. Estas dos versiones simultáneas fueron, una basada en hilos y la otra basada en **eventos**, yo empecé a trabajar en la versión basada en hilos mientras que un compañero de área se encargó de realizar la versión basada en eventos.

Imagen 3.3 Segunda versión cliente de video vigilancia

Las capacidades de esta versión también crecieron, ahora el *software* se volvió más dinámico ya que el cliente podía agregar o quitar cámaras cuando se requiriera, por ejemplo en caso de que la conexión a dicha cámara se perdiera o cuando simplemente ya no se quisiera continuar visualizando una cámara y se deseara ver otra. También fueron solucionados algunos problemas de video, por ejemplo el problema relacionado a la repetición de los *frames* y lo solucioné de la siguiente forma, después de revisar un poco el código del proyecto, me di cuenta que una de las llamadas a los métodos utilizados por el *software* era realizado dos veces, lo que provocaba el efecto de repetición del *frame* que se comentó anteriormente, pero en realidad esta repetición nunca sucedía, lo que si pasaba era que los *frames* se iban apilando según como llegaran (dependían de los hilos) y había ocasiones en que llegaba primero un frame que otro y por eso ocurría ese problema. Para resolverlo solo basto con quitar una de las llamadas a ese método y el sistema empezó a mostrar correctamente el video.

En esta versión se incorporó un poco de organización al proyecto, tanto en la parte grafica como en la organización de los directorios que conformaban la aplicación. Las partes principales de la organización grafica eran tres:

1. El menú superior
2. El listado de cámaras
3. La visualización de las cámaras

Menú superior

Este menú superior contaba con ocho botones:

- Abrir configuración
- Guardar configuración
- Agregar cámara
- Vista Default
- Vista 5x4
- Vista 4x4
- Vista 4x3
- Vista 3x3

Y por supuesto contaba también con los botones de minimizar, restaurar y cerrar que cualquier otro *software* de escritorio tiene. De los cuales el botón de cerrado tenía una función extra aparte de la de cerrar el programa. La cual será comentada en la siguiente descripción de cada uno de los botones.

 Guardar configuración.

Una de las capacidades agregadas al *software* en su segunda versión fue la de guardar un archivo de configuración, el cual contenía información de las cámaras que se deseaban guardar para que en otro momento se utilizara ese mismo archivo para poder agregar y visualizar de manera automática las cámaras guardadas, esta fue una gran característica agregada ya que sin ella el usuario de la aplicación tendría que haber agregado las cámaras que quisiera visualizar una a una y lo tendría que hacer cada vez que se ejecutara el programa. Con el archivo de configuración esta actividad se tenía que realizar solo una vez para posteriormente solo utilizar el archivo de configuración.

La idea de contar con este archivo de configuración surgió con el propósito de que el usuario pudiera por ejemplo guardar cámaras por colonias o avenidas y así en el momento deseado podría abrir el archivo de configuración que necesitase y entonces las cámaras se cargarían automáticamente.

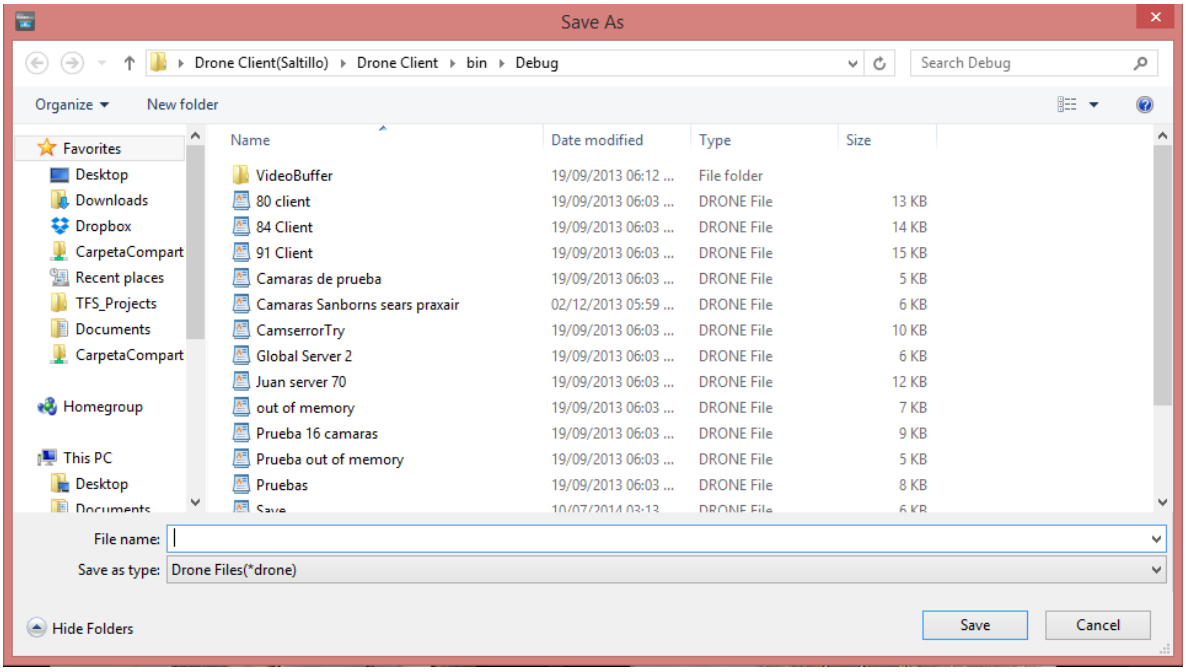


Imagen 3.4 Explorador de Windows para guardar configuración

Al momento de pulsar este botón se abría un explorador de Windows (Imagen 3.4), mismo que permitía proporcionar el nombre con el que se quería guardar el archivo de configuración. Internamente, al nombre proporcionado por el usuario, se le agregaba la extensión ".drone", esto con el propósito de identificar los archivos de configuración que se fueran generando, después se utilizaba una lista de *strings* que contenía la dirección de las cámaras que se estaban visualizando, esta lista era modificada agregando el nombre de la pestaña que contenía a las cámaras formando una lista de *strings* parecida a la siguiente.

Pestaña 1
URL cámara 1
URL cámara 2
…
Pestaña 2
URL cámara 1
URL cámara 2

Esta nueva lista era serializada para generar el archivo de configuración de cámaras. En esta versión el usuario era libre de guardar el archivo antes mencionado en el directorio que más le pareciera adecuado.

 Abrir configuración.

Por supuesto que al existir una funcionalidad para poder guardar las cámaras, también tenía que existir una funcionalidad que permitiera abrir un archivo de configuración en el momento que fuera necesario, la imagen mostrada arriba corresponde al botón de la funcionalidad descrita. Lo primero que ocurría al pulsar este botón era preguntarle al usuario si deseaba conservar las cámaras que ya estaban en reproducción o si quería quitarlas para cargar solo el archivo de configuración seleccionado (Imagen 3.5).

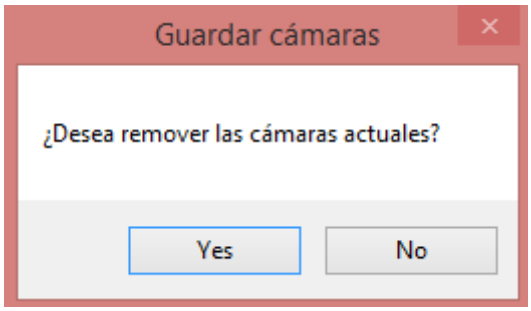


Imagen 3.5 Remover cámaras actuales

Internamente si el usuario optaba por la opción de remover las cámaras, lo único que se tenía que hacer era limpiar una lista que llevaba el control de las cámaras que se iban agregando, de lo contrario se respetaba el contenido de la lista. Una vez terminada esta acción, se le mostraba al usuario un explorador de Windows, el mismo explorador que se le mostraba cuando se hacia el intento de guardar una configuración de cámaras, solo que en esta ocasión por supuesto se tenía que elegir el archivo que se quería cargar.

Una vez que el usuario seleccionaba el archivo de configuración que quería utilizar para cargar cámaras, el archivo seleccionado era de-serializado formando una lista de *strings* como la siguiente:

Pestaña 1
URL cámara 1
URL cámara 2
…
Pestaña 2
URL cámara 1
URL cámara 2
…

Entonces para cargar de forma automática las cámaras contenidas en esta lista de *strings*, se realizaba un ciclo en el que se comparaba si el *string* analizado era una *URL* valida, en caso contrario ese *string* era considerado como nombre de pestaña, de esta forma se tenía conciencia de cuando se tenía que generar o construir una pestaña y cuando se tenía que construir una cámara y también se sabía a qué pestaña correspondía dicha cámara. El *string* que no correspondía a una cámara era puesto como nombre de la pestaña que contenía a las cámaras.

Agregar Cámara

El tercer botón de izquierda a derecha que aparece en el menú superior tenía la finalidad de proporcionar un control de usuario que permitía realizar la configuración de las cámaras para que estas fueran agregadas a la interfaz de usuario y pudieran ser visualizadas. El control de usuario que se presentaba al pulsar el botón, estaba formado por un elemento del diseñador de *Visual Studio* llamado *TabControl* el cual permitía agregar pestañas y cada pestaña podía tener su propia funcionalidad, en el caso de este control de usuario, se generaron tres pestañas:

- *Camera*
- *Stream*
- *Player*

Camera

Esta pestaña (Imagen 3.6) estaba diseñada para que el usuario proporcionara datos tales como la *IP* de la cámara, la resolución, la velocidad de cuadros por segundo que la cámara iba a tener, la compresión, el método de entrega utilizado en la sesión establecida entre el cliente y el servidor. En caso de que la cámara contara con autentificación para poder visualizar su video, este control de usuario también permitía introducir los datos necesarios para poder utilizar dicha cámara.

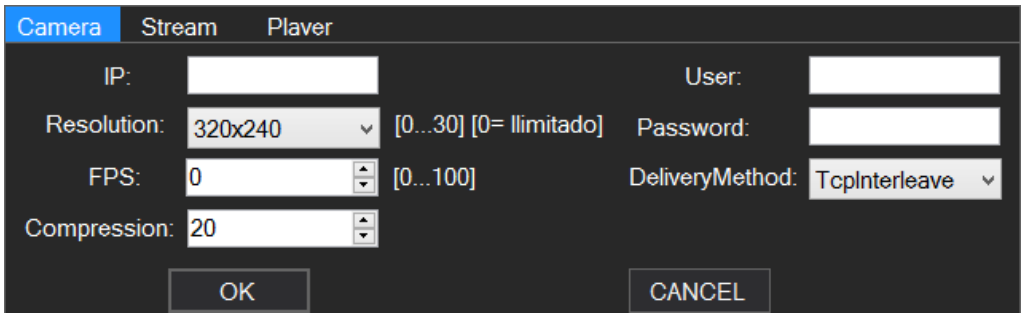


Imagen 3.6 Formulario Camera

Al pulsar el botón “OK” que aparece en la imagen anterior, el programa tomaba los valores *IP*, *Resolution*, *FPS* y *Compression* para generar la *URL* con la cual se intentaba levantar la sesión hacia la cámara propietaria de la dirección *IP*, después de esto se utilizaba la otra solución que formaba a este proyecto, la cual será descrita más adelante y para ello fue utilizado el parámetro *DeliveryMethod* que especificaba que método de entrega era utilizado para transmitir el video en este caso desde la cámara hacia el cliente. Una vez configurada la *URL* y establecida la sesión, el usuario podía empezar a visualizar el video, en caso de que este hubiera sido correctamente inicializado.

Stream (Imagen 3.7)

Esta pestaña estaba pensada para que el usuario pudiera realizar conexiones hacia el servidor, y pedirle a este que le brindara el flujo de video que necesitase. En este caso el formulario disminuía la cantidad de datos que tenían que ser proporcionados por el usuario, ya que la magia de realizar las configuraciones se hacían en la parte del servidor. En este caso eran solo cuatro parámetros.

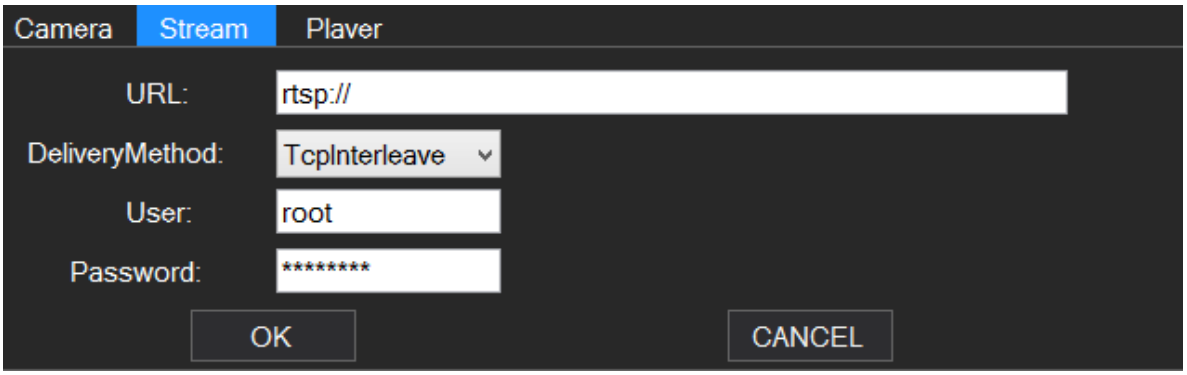


Imagen 3.7 Formulario Stream

El primer parámetro correspondía a la *URL* de alguna de las cámaras que el servidor tuviera levantadas, solo se tenía que agregar la dirección del servidor y el identificador de la cámara para indicarle al servidor lo que queríamos ver. Era algo como esto rtsp://X.X.X.X/Camara1, después de esto y como en la pestaña anterior, se tenían que indicar tanto el método de entrega, como un usuario y una contraseña. El usuario y la contraseña en este caso eran utilizados en caso de que la cámara fuera especial, por ejemplo, que tuviera la capacidad de moverse, para lo cual solo usuarios que tuvieran esta contraseña serían los autorizados a realizar movimientos a la cámara.

Player

La pestaña *Player* (Imagen 3.8) en esta versión de *software* la única capacidad que tenía era la de reproducir videos que cumplieran con el formato H.264 y que estuvieran almacenados localmente, pero también se quería extender esa funcionalidad para elegir una fecha de inicio y una fecha de finalización del video indicando en ambos casos la hora en la que se quería ver y dejar de ver el video bajo demanda.

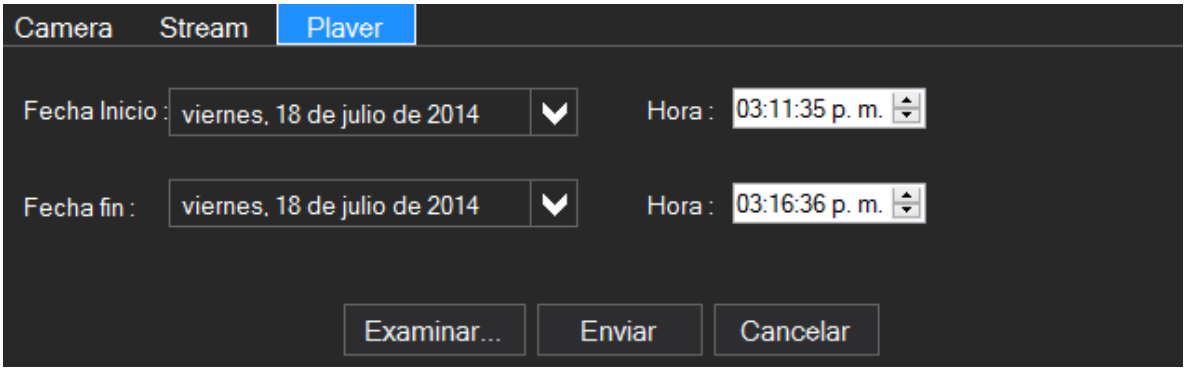


Imagen 3.8 Formulario Player

Cuando se pulsaba el botón (flecha hacia abajo) de las fechas de inicio y fin se mostraba un calendario flotante personalizado que permitía elegir el día que deseábamos. Al pulsar el botón de enviar internamente se generaría una cadena la cual sería codificada y enviada al servidor, éste la decodificaría y obtendría los datos que el cliente había proporcionado y entonces el *stream* sería enviado al cliente para que este pudiera ver el video grabado.

Ahora bien cuando el botón de examinar era pulsado lo único que ocurría era que se abría nuevamente el explorador de Windows en el directorio en donde estaban almacenados los videos locales y entonces el usuario podía elegir el video para empezar a visualizarlo.

Vista Default

La vista que venía como predeterminada en este *software* era una formada por una matriz de cuatro renglones por cinco columnas (Imagen 3.9), la cual tenía la particularidad de que mientras se iban agregando cámaras, ésta se iba redimensionando, es decir al haber una sola cámara, dicha cámara ocupaba todo el espacio disponible de la vista *default*, y mientras se iban agregando más cámaras, éstas se repartían el espacio disponible para ser ubicadas. Por lo que conforme había mayor cantidad de cámaras, el tamaño de éstas iba disminuyendo.

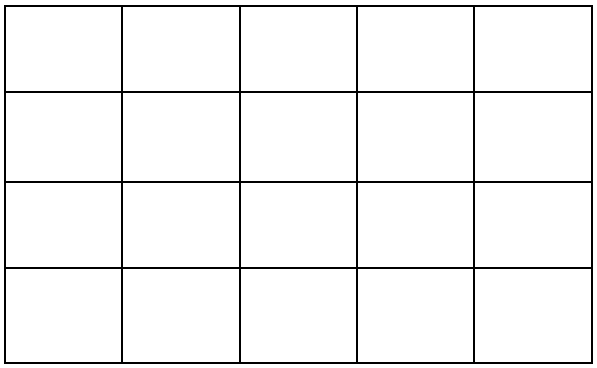


Imagen 3.9 Vista Default

Cada cuadro mostrado en la parte de arriba, era un control de usuario llamado *VideoDisplay*, entonces cuando el cliente iba agregando las cámaras al lienzo, lo que hacía primero internamente era contar la cantidad de cámaras que había en la vista *default*, para después enviar esta cantidad a un *switch* mismo que ya tenía una numerosa cantidad de casos en los que dependiendo de la cantidad de cámaras recibida, devolvía las dimensiones y posición que debía tomar cada uno de los *VideoDisplay* que contenían a las cámaras. Este cálculo se hacía cada vez que una cámara era agregada y también cada vez que una cámara era eliminada.

Vista 5x4

Esta y las demás vistas tenían casi las mismas capacidades, el aspecto en el que principalmente cambiaba estaba en el aspecto visual ya que como se ve en la siguiente imagen este permitía posicionar o mejor dicho tener una cámara con mayores proporciones en el centro para que el usuario pudiera ver más a detalle lo que estaba ocurriendo en la escena visualizada por la cámara.

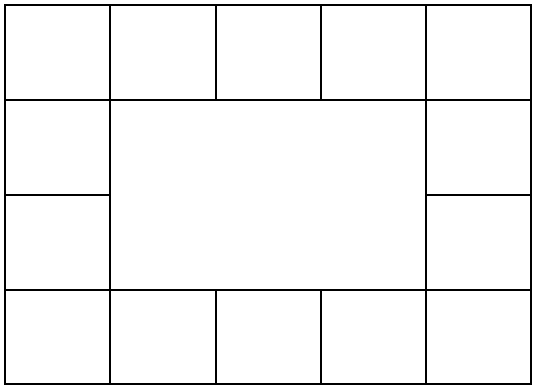


Imagen 3.10 Vista 5x4

En este caso también se podía arrastrar cualquiera de las cámaras exteriores hacia la cámara o el lienzo que se encontraba en la zona central de esta vista, pero al tratar de arrastrar la vista central no se presentaba el efecto de intercambio de cámaras, en lugar de eso lo que ocurría era que la cámara central cambiaba de posición generando las siguientes vistas.

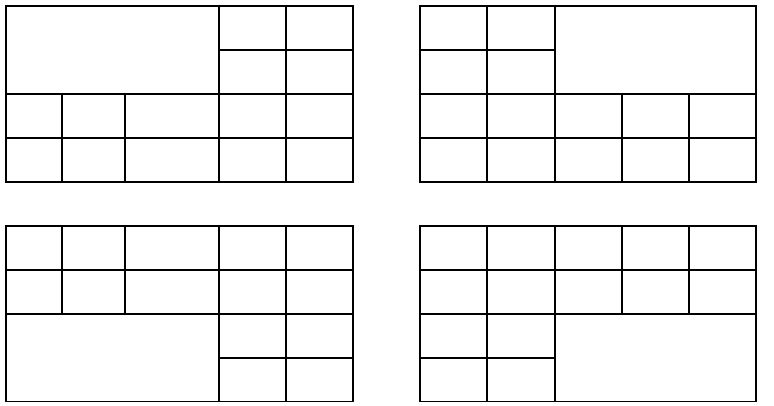


Imagen 3.11 Ordenamiento de cámaras en Vista 5x4

Otra de las diferencias era la forma en la que se iban posicionando las cámaras cada vez que una era agregada, ésta vez en lugar de ocupar el espacio total en donde se visualizaban las pantallas, cada cámara agregada tenía un tamaño y posición fija, el cual empezaba en la esquina superior izquierda, de tal manera que hasta la séptima ocasión que se agregaba una cámara, se visualizaba el *VideoDisplay* central.

Vista 4x4

La vista de 4x4 (Imagen 3.12) era capaz de soportar 8 cámaras visualizándose al mismo tiempo, contaba con las mismas capacidades que las vistas anteriores y al igual que en la vista 5x4 al arrastrar el *VideoDisplay* más grande este cambiaba de posición pero no intercambiaba su contenido, lo que si ocurría cuando se arrastraba un *VideoDisplay* de los pequeños hacia la parte central.

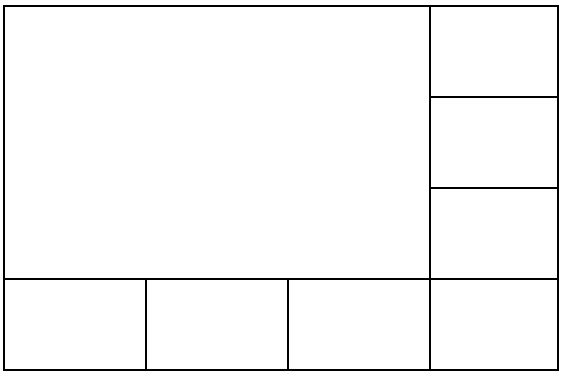


Imagen 3.12 Vista 4x4

Vista 4x3 y 3x3

Las vistas tienen las mismas capacidades en cuanto a funcionalidad, solamente variaba la cantidad de cámaras que cada una de ellas podía mostrar al mismo tiempo, la vista 4x3 era capaz de visualizar 7 cámaras, mientras que la vista 3x3 era capaz de visualizar un total de 6 cámaras (imagen 3.13).

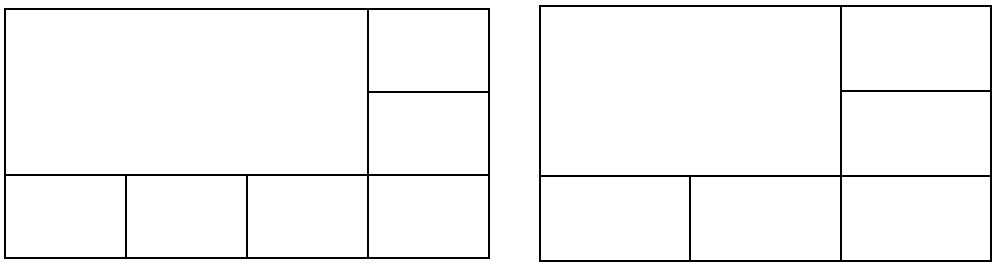


Imagen 3.13 Vistas 4x3 y Vista 3x3

Ahora bien en un principio las vistas solo podían ser seleccionadas en caso de que no pasaran la cantidad máxima que cada vista podía soportar, esto es, si se habían agregado 7 cámaras no podía ser seleccionada la vista 3x3 con el propósito de no perder de vista alguna de las cámaras. Aunque esta decisión de permitir la selección de vistas solo si no se rebasaba el límite parecía una buena idea, mi jefe directo me pidió que diera la libertad de escoger cualquier vista sin importar que se perdiera alguna cámara.

Visualización de las cámaras

Para poder mostrar las cámaras, se creó un control de usuario llamado *TabVideoWall* el cual permitía llevar el control de la creación de pestañas en el formulario principal, de las vistas que podía soportar, y de una característica que le agregaba atractivo visual al *software*, la cual se trataba del arrastre de las cámaras e intercambio entre ellas. La estructura de este control se puede apreciar en la siguiente imagen.

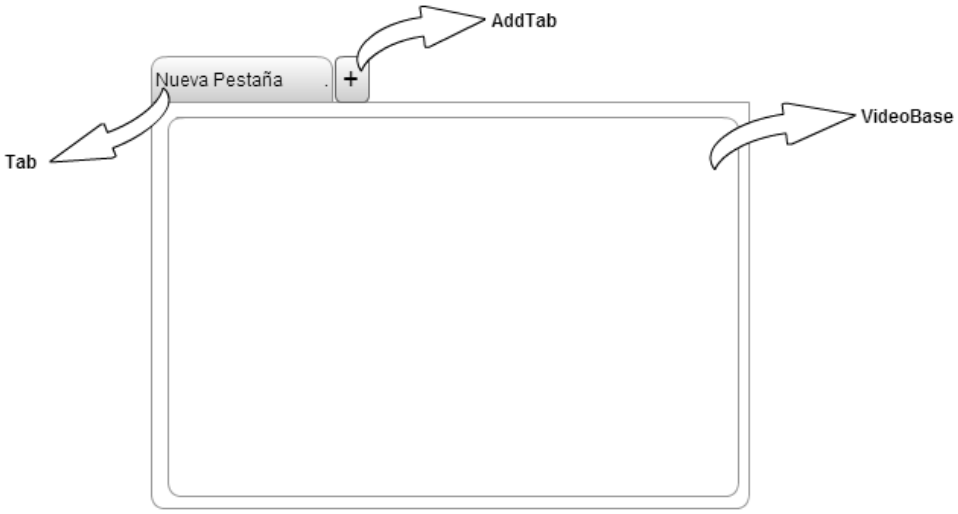


Imagen 3.14 TabVideoWall

Cuando iniciaba el programa, se construía un control de este tipo e internamente en este control se creaba una pestaña (*Tab*), ya que como mínimo este control debía de tener una pestaña para poder contener la vista en la cual se podían agregar las cámaras, y también se agregaba un botón llamado *AddTab* ya que sin éste el programa solo se limitaría a mostrar veinte cámaras. El nombre predeterminado del cada *Tab* era "Nueva Pestaña" y al hacer doble clic sobre la pestaña, el usuario podía cambiar este nombre. Ahora bien el botón *AddTab* simplemente daba la capacidad como su nombre lo indica, de agregar más pestañas al control. Mientras que en el *VideoBase* lo que se hacía era contener las vistas disponibles en la interfaz de las cuales se habló anteriormente.

Cada una de las vistas estaba conformada por conjuntos de *VideoDisplay* en los cuales era en los que se podían visualizar las cámaras. Cuando se quería realizar el arrastre de uno de estos *VideoDisplay*, ocurrían varios eventos, estos ocurrían tanto en el *VideoDisplay* como en el padre o contenedor del *TabVideoWal*, los eventos ocurridos eran:

- *MouseDown*
- *MouseMove*
- *GiveFeedback*
- *MouseUp*
- *DragEnter*
- *DragDrop*

De estos seis eventos, los que eran desencadenados en el *VideoDisplay* eran *MouseDown*, en el cual ocurrían dos acciones diferentes dependiendo de la cantidad de veces que se presionara el botón izquierdo del *Mouse*. En caso de que se presionara dos veces lo que se hacía era que sobre la interfaz de usuario aparecía un control de usuario (Imagen 3.15), cuya funcionalidad ya fue descrita cuando se habló de del botón “Agregar Cámara”.

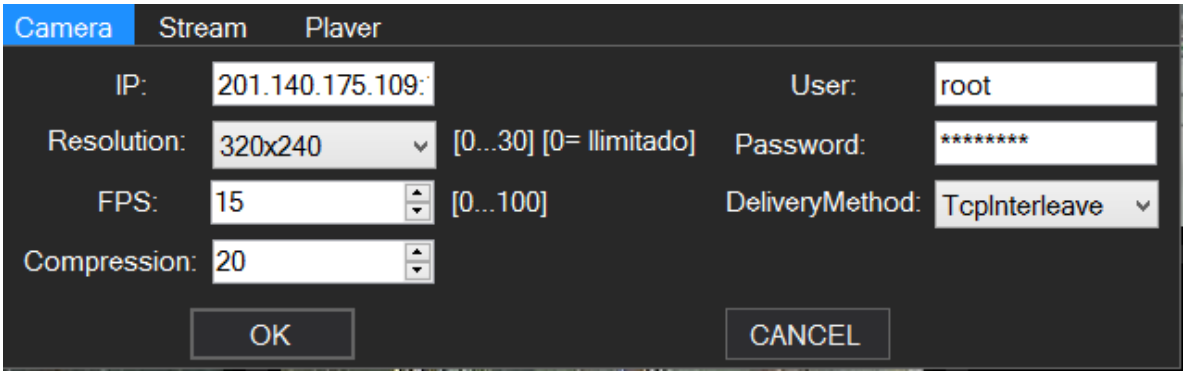


Imagen 3.15 Formulario “Agregar cámara”

Ahora bien si el botón era pulsado una sola vez y así se mantenía lo que se hacía era indicarle al *VideoDisplay* que se le iba a dar la capacidad de moverse, para que éste permitiera comenzar el arrastre de la cámara de un lugar a otro. Una vez que este evento ocurría, se generaba un nuevo control de usuario que lo único que hacía era contener una copia del video que se estaba visualizando y que se había empezado a arrastrar. Este control de usuario tenía que saber el momento en el que el mouse generaba un movimiento para que éste lo siguiera por todo el espacio disponible, Esto ocurría ya en la parte del *TabVideoWall*, más específicamente en el evento *MouseMove*. Pero un inconveniente de esta forma de implementar el evento fue que aunque el arrastre ocurría, el mouse no se veía dentro del control que estaba siendo arrastrado (Imagen 3.16).

Imagen 3.16 Arrastre de cámara

Ahora bien para que el video siguiera transcurriendo mientras el *VideoDisplay* estaba siendo arrastrado, se utilizaba el evento *GiveFeedback*, en el cual se iba actualizando el video para que se viera la transmisión tanto en el control que estaba siendo arrastrado, como en el *VideoDisplay* origen.

Cada vez que iba saliendo o entrando en algún elemento de la vista, ocurría el evento *DragEnter* con el que se podía obtener la posición en la que se encontraba el control de usuario que daba el efecto de estar siendo arrastrado, para que una vez que se decidiera colocar la cámara (*VideoDisplay*) en otra ubicación se pudiera realizar el cambio de posición de dicha cámara, esto ocurría cuando el usuario soltaba el botón del mouse (evento *MouseUp*) con lo cual ya se conocían el origen y el destino de la cámara y entonces se ejecutaba el evento *DragDrop* para poder realizar el cambio.

Este cambio en realidad era un cambio de referencia, es decir, el control que contenía a las cámaras (*VideoDisplay*) nunca se movía, lo único que se hacía era cambiar la referencia de hacia donde se estaba dirigiendo el video. En la parte de la visualización de las cámaras aparte del modo en el que se ordenaban y de los eventos que ocurrían durante el arrastre de alguna cámara, también se podía hablar de otros detalles importantes que se implementaron en esta versión. Uno de ellos era que en el momento en el que se trataba de hacer conexiones a las cámaras, en el *VideoDisplay* se mostraba durante cierto tiempo el estado de la conexión, estos estados eran tres, el de conectando mismo que se identificaba con un pequeño indicador amarillo, el de conectado al cual le correspondía un indicador verde, además de que el video ya empezaba a visualizarse y el de sin conexión el cual tenía un indicador rojo y un mensaje que le indicaba al usuario justamente eso, que el flujo de video no estaba disponible.

Cuando el usuario intentaba realizar alguna conexión el indicador amarillo empezaba a parpadear (imagen 3.17) hasta que ocurría alguna de las dos situaciones descritas en el párrafo anterior.

Imagen 3.17 Indicador amarillo

Si la conexión se lograba, durante algunos segundos se mostraba el indicador verde (imagen 3.18), éste aparecía sobre el vídeo.

Imagen 3.18 Indicador verde

En caso de que la conexión no se lograra establecer se mostraba el indicador rojo (imagen 3.19) hasta que el usuario tomara acciones para volver a conectarse o para eliminar la cámara.

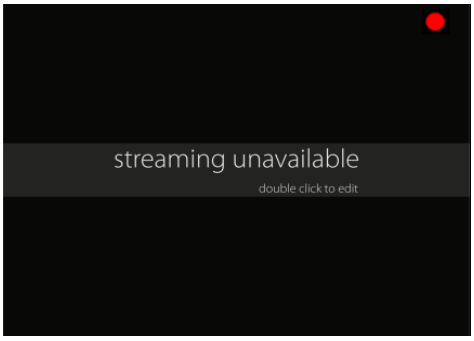


Imagen 3.19 Indicador rojo

De esta forma el usuario podía identificar rápidamente el proceso que estaba llevando a cabo el cliente de vídeo para realizar una nueva conexión, pero cuando se tenían varias cámara a las cuales se estaba intentando conectar, el cliente de vídeo parecía una especie de árbol de navidad ya que todas las cámaras se encontraban parpadeando al mismo tiempo, lo cual se tornaba en un efecto visual bastante molesto para quien estuviera haciendo uso del software. Aparte de estos indicadores, al pulsar el botón derecho del mouse, se mostraba un pequeño menu con cuatro opciones:

- *Remove*
- *FullScreen*
- *Refresh*
- *EditSource*

Remove obviamente eliminaba la cámara sobre la cual se había efectuado el clic derecho, *FullScreen* mostraba en pantalla completa la cámara, lo cual no se podía lograr de forma directa ya que el *VideoDisplay* se encontraba insertado en una celda de la vista, para lograrlo, detrás de la vista se encontraba otro *VideoDisplay* del tamaño de la vista, hacia el cual se enviaba el video en el momento en el que se presionaba la opción de *FullScreen*. El botón de *Refresh*, intentaba reconectarse a la cámara, para realizar esto, primero intentaba terminar la sesión que se tenía con la misma para liberar la mayor cantidad de recursos posibles y después intentaba cargar una nueva conexión con dicha cámara.

Al pulsar en el botón *EditResource*, el programa mostraba el control de usuario que ya se comentó anteriormente, el cual se utilizaba también cuando el usuario intentaba agregar una nueva cámara para que pudiera ser visualizada, después de que la configuración había sido realizada, se cerraba la conexión actual que se tenía con la cámara para liberar los recursos y eliminar todas las relaciones que se tenían con dicha cámara, y una vez eliminada todo tipo de relación, se generaba una nueva instancia que se cargaba en el mismo *VideoDisplay* que ya existía.

Algunas de estas acciones también se podían realizar sin la necesidad de hacer clic derecho sobre el *VideoDisplay* ya que dicho *VideoDisplay* también contaba con tres botones que se mostraban en todo momento siempre y cuando se estuviera transmitiendo vídeo, estos botones eran:

- *Remove*
- *Fullscreen*
- *SnapShot*

Los primeros dos botones utilizaban el mismo manejador de eventos que el utilizado al hacer clic derecho y tratar de realizar las acciones ya descritas; mientras que el tercer botón (*Snapshot*) tomaba una captura del vídeo deseado, cuando se realizaba esta acción por primera vez, el programa se encargaba de crear una carpeta dentro del directorio "Documentos" con el mismo nombre de este *software* y, también, si se estaba realizando transmisión del vídeo desde el servidor, se creaba un directorio con el nombre que la cámaras eran identificadas (Imagen 3.21) y en ese directorio era donde se guardaba la imagen que había sido capturada.

Imagen 3.20 Estructura de directorios para capturas de pantalla

En el momento que se iniciaba una conexión o en caso de que la conexión no fuera exitosa el único botón que se mostraba era el de eliminar cámara, por otro lado cuando la cámara se encontraba en *fullscreen* el botón de eliminar no aparecía. El *VideoDisplay* también se encontraba suscrito a un evento que era llamado en el momento en el que el mouse entraba en él mismo, esto con el objetivo de mostrar un marco de color azul alrededor del *VideoDisplay*, el cual servía para darle énfasis a la cámara que se estaba visualizando, este indicador convive también con el listado de las cámaras cuya relación será comentada más adelante, cuando se hable sobre este listado de cámaras.

Listado de cámaras

El listado de las cámaras se llevaba a cabo dentro de otro control de usuario llamado *NavigateBar*, el cual era utilizado para poder mostrar y ocultar la lista de cámaras que eran agregadas para visualización, cuando el usuario del programa así lo quisiera, la decisión de ocultar la lista de cámaras fue con el objetivo de permitir que el usuario tuviera mayor espacio de visualización. En su forma oculta la instancia de la clase *NavigateBar* solamente se mostraba como un botón (Imagen 3.22), mientras que cuando se presionaba este botón se mostraba el listado de las cámaras (imagen 3.23) que se estaban reproduciendo en ese momento así como también las pestañas que se habían creado mientras se utilizaba el cliente de video.

Imagen 3.22 Botón lista de cámaras

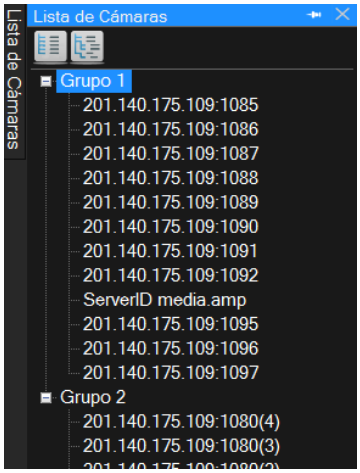


Imagen 3.23 Listado de cámaras

En la parte superior de la instancia del *NavigateBar* se encontraban dos botones, mismos que permitían por medio de un evento, colapsar y expandir la lista de cámaras contenida en cada pestaña. Una de las particularidades que tenía este listado de cámaras era que el usuario podía seleccionar de la lista mostrada en la imagen anterior alguna de las cámaras y esta automáticamente iba a ser resaltada en la parte de la visualización de las cámaras haciendo usos del borde que tenía cada *VideoDisplay* el cual cambiaba a color azul, para que así el usuario pudiera encontrar rápidamente una cámara.

Esta característica se llevaba a cabo por medio del evento *MouseDown* del elemento que conformaba el listado (que en este caso era un Sting con la descripción de la cámara) y lo único que hacía era eso cambiar el color del borde del *VideoDisplay*. La capacidad de encontrar la cámara al hacer clic en el listado funcionaba aun cuando la cámara en la que se hacía dicho clic no estaba en la pestaña que estaba actualmente mostrándose.

Además de este evento el *NavigateBar* tenía asociados algunos otros, como por ejemplo en caso de que quisiéramos que este listado estuviera permanente en la pantalla se podía pulsar un botón en la parte superior del *NavigateBar*, el cual tenía como imagen descriptiva un pin que cambiaba su posición dependiendo de si el *NavigateBar* iba a quedarse fijo en la interfaz o si iba a desaparecer. En caso de que el usuario quisiera dejar permanente el listado de cámaras, se realizaba un redimensionamiento de los controles que se encontraban en la interfaz, en este caso los controles que principalmente cambiaban eran el *NavigateBar* que quedaba fijo en la interfaz y el *VideoBase*, pero como el *VideoBase* tenía contenida la vista y la vista tenía contenidos a los *VideoDisplay*, todos estos controles tenían que cambiar de tamaño y posición para poder hacerle espacio al *NavigateBar* para que pudiera quedar fijo en la interfaz. Todos estos redimensionamientos y cambios de posición se tenían que hacer por código duro, ya que no había una forma automática de que pudieran realizarse. Una vez que se realizaba todo este procedimiento, el aspecto del programa era el siguiente.

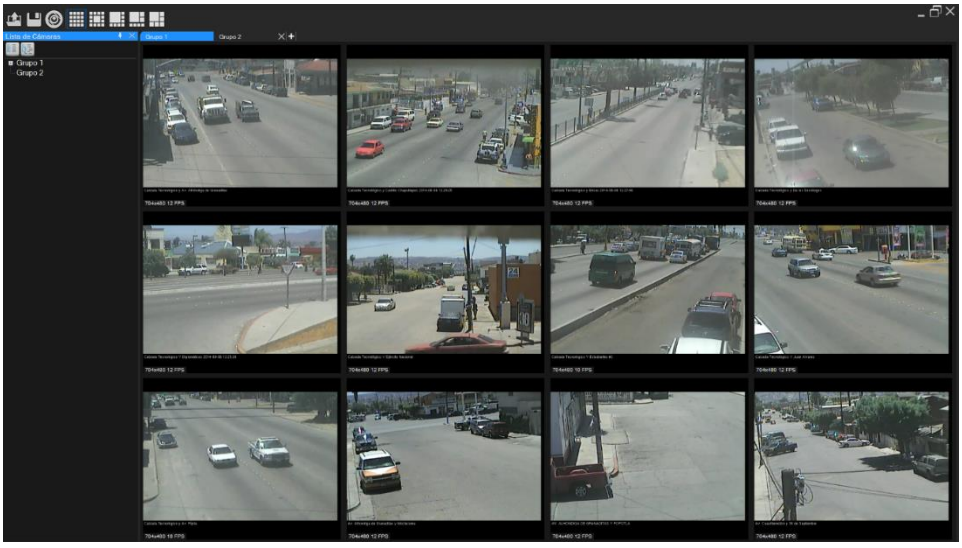


Imagen 3.24 NavigateBar fijo

En el que se puede observar como todas las cámaras son visibles al usuario, cosa que no ocurría cuando el *NavigateBar* no estaba fijo en la interfaz, dicha condición provocaba que este control de usuario se encimara en las cámaras y no le permitiera al usuario visualizar algunas de ellas mientras este control estuviera mostrándose. Para poder regresar a la vista en la que el listado de cámaras aparecía como un botón, simplemente se tenía que pulsar el botón "x" del *NavigateBar*.

Una vez comentadas todas la funcionalidades de esta versión hablaré ahora de los problemas de la misma basándome primero en las cuestiones relacionadas al video y después en la secuencia de descripción que hice para las funcionalidades, pero antes, quiero mencionar una de la capacidades importantes que también se agregó a esta versión y esta surgió cuando noté que el hecho de estar ejecutando la aplicación y después cargar el archivo necesario para ver las cámaras se tornaría en una actividad tediosa para el usuario, por lo que hice que esta versión fuera capaz de guardar automáticamente las cámaras que se estuvieran visualizando en el momento en el que el usuario decidiera cerrar el programa, de este modo cuando el usuario volvía a ejecutar el programa y a iniciar sesión en el mismo, dichas cámaras se cargaban automáticamente. Este archivo fue nombrado por default "Save" y por supuesto se manejó la misma extensión que para los archivos que tenía que guardar el usuario si así lo quisiera.

Bugs de la versión 2

En esta versión el *software* ya fue capaz de soportar mucho más de cuatro cámaras, pero los bugs de la versión anterior continuaban, es decir se seguían viendo en ocasiones dos videos en un mismo *VideoDisplay*, aunque en esta ocasión supongo que porque se mejoró un poco el rendimiento del programa este problema ocurría con menor frecuencia, pero aun ocurría y no se conocía todavía la razón por la que esto pasaba. Además de este problema, se generó uno nuevo el cual consistía en que el video parecía no refrescarse y quedaban fantasmas de las personas u objetos que se movían en el área de visión de la cámara. Ahora bien en cuanto a las cuestiones relacionadas con la interfaz y la interacción con el usuario, también se presentaron muchos errores tanto visuales como errores que hacían que el programa fallara y se tuviera que reiniciar.

Empezaré hablando de los botones del menú, este problema fue más que nada visual ya que al pulsar tanto el botón de guardar configuración como el de abrir configuración, la interfaz de las ventanas que aparecían después de ser pulsados eran demasiado contrastantes con la interfaz del programa que desarrollé, esto debido a que se utilizó el propio sistema de archivos de Windows para dar solución a estas dos acciones y la interfaz por *default* de los controles de Windows es demasiado pobre.

Otro de los aspectos que no me pareció correcto aunque en su momento así se dejó, fue la lógica del guardado de las cámaras ya que se hizo demasiado básico, porque no se tenía como tal en ese momento una forma de indicar qué cámaras pertenecían a qué pestaña ya que estos dos campos se guardaban de la misma manera, es decir, como *string* y de igual forma al momento de querer abrir alguna configuración se realizaba de una forma errónea, por medio de comparaciones para saber si era una cámara y en caso de que no fuera así, se daba por hecho que se trataba de una pestaña.

Los problemas relacionados al agregar cámaras eran que cualquier usuario que hiciera uso del programa tenía que estar agregando las cámaras a mano y este llenado de cámaras era demasiado pesado ya que tenía demasiados campos que se tenían que llenar una y otra vez cada que el usuario del programa quisiera agregar una nueva cámara y por ejemplo si la cámara tenía contraseña cualquier usuario tenía que conocerla para que pudiera lograr la conexión a la misma, lo cual implicaba un problema de seguridad.

Ahora bien, las vistas eran las que tenían la mayor cantidad de errores junto con el *VideoDisplay*, estos errores provocaban un alto procesamiento del programa lo que hacía que demandara demasiados recursos, tantos que con tan solo 20 cámaras el programa dejaba de funcionar. Pues bien las vistas demandaban tales recursos para poder hacer cambios entre las mismas ya que se tenía que estar redimensionando demasiado cada que se hacía un cambio además de que en el momento en el que se realizaba el cambio se volvían a crear cada uno de los *VideoDisplay* para poder redimensionarlos y posicionarlos adecuadamente.

La demanda de recursos en la vista también ocurría cuando se anclaba el listado de cámaras para que estuviera siempre visible, lo cual provocaba que se redimensionara la vista y que se redimensionaran también los *VideoDisplay* contenidos en la vista, y al no realizarse de forma automática, se tenían que poner demasiadas líneas de código para lograr realizar este procedimiento ya que dependiendo de la vista el *software* entraba en un *switch* que tomaba decisiones de cómo se tenían que ordenar las cámaras que una vez que ya habían sido agregadas a la vista y dependiendo del número de cámaras se entraba a otro *switch* que especificaba el tamaño de los *VideoDisplay*. Cada vista tenía prácticamente el mismo código, solo variaba en ciertas partes lo que en su momento hacia que como programador no estuviera reutilizando la mayor cantidad de código posible.

Otro de los aspectos que también demandaba bastantes recursos y que tenía demasiadas líneas de código era en el momento en el que se deseaba arrastrar la cámara de un lado a otro ya que por cada vista se creaba un *switch* solamente para conocer la posición en la que había quedado nuestro *VideoDisplay*. En el *VideoDisplay* en esta versión, la reconexión se había realizado de forma errónea ya que se había utilizado recursividad para realizar la reconexión, pero si la cámara no llegaba a responder, después de cierto tiempo el programa fallaba ya que durante cada intento de reconexión, se intentaba establecer una sesión misma que no había sido controlada para que se liberara la memoria y al tratarse de un método recursivo esta memoria se iba acumulando y por ende este fallo ocurría.

El *VideoDisplay* también era demasiado agresivo para la vista de los usuarios cuando se estaba intentando realizar la conexión a las cámaras ya que como se había dicho, éste mostraba un pequeño indicador de color que le decía al usuario el estado de la conexión, pero este indicador, cuando estaba en color amarillo, es decir, cuando se estaba intentando realizar la conexión a la cámara, parpadeaba y al estar veinte cámaras con el indicador parpadeando constantemente, molestaba la vista del usuario.

Ahora bien, cuando se perdía la conexión de la cámara o cuando no se podía realizar la conexión con la misma, solo se mostraba un mensaje al usuario, el cual no tenía ninguna funcionalidad extra que le permitiera al usuario tomar alguna decisión.

Otro indicador que en ocasiones fallaba era el contorno azul que se dibujaba alrededor del *VideoDisplay* cuando el puntero del mouse estaba sobre éste ya que si el mouse abandonaba el *VideoDisplay* por ciertas áreas, el contorno azul no desaparecía y al pasar el puntero por otro *VideoDisplay* también se dibujaba el contorno azul por lo que daba la sensación de tener seleccionados dos *VideoDisplay* al mismo tiempo. Otro *bug* ocurría cuando se intentaba cambiar la configuración del *VideoDisplay* para cambiar la cámara que se estaba visualizando, ya que en caso de cancelar este cambio de configuración el *software* reiniciaba la conexión a la cámara lo cual hacia que se volvieran a consumir recursos en lugar de mantener los que ya se estaban utilizando.

En cuanto a las pestañas, se tuvo que realizar un *TabControl* personalizado para agregarle un botón a dicho control para poder añadir más pestañas a la interfaz pero no se había controlado la situación en la que hubiera demasiadas pestañas abiertas al mismo tiempo, ya que si esto ocurría, se perdían las pestañas que ya no cabían en la vista contenedora y aunque también se habían agregados botones que recorrían cada pestaña, visualmente no era muy agradable ya que los controles se encimaban.

Dentro de la pestaña cuando se intentaba cambiar su nombre, no era muy claro el momento en el que el usuario podía realizar el cambio e incluso podía dejar el nombre en modo de edición y seguir utilizando el *software* sin que se completara la acción de cambio de nombre. La eliminación de la pestaña también creaba conflictos, ya que en ciertas ocasiones cuando se realizaba este eliminado de pestaña y después se agregaba una nueva, las cámaras que estaba contenidas en la pestaña que se habían “eliminado”, volvían a aparecer en la nueva pestaña, lo cual no debía ocurrir.

Al tratarse de un *software* de video vigilancia lo principal era eso, poder visualizar en todo momento el video que le interesaba al usuario, pero como se vio en la descripción de la funcionalidad del listado de cámaras, éste listado se encimaba en las cámaras que se encontraban del lado izquierdo obstruyendo la visibilidad de las mismas afectando el desempeño del personal que estuviera haciendo uso del *software*.

Para terminar con esta versión, solo cabe mencionar que aunque se intentó y de cierto modo se logró darle un poco de organización al proyecto de equipo en el que participe, todavía seguía manteniendo irregularidades ya que las cuestiones que eran meramente relacionadas al video, estaban combinadas con aspectos de la interfaz gráfica lo cual no era correcto. En la versión que se describe a continuación se lograron reparar la mayoría de los *bugs* que esta versión dejo, así como también se logró que visualmente tuviera un mejor aspecto.

Tercera versión

Imagen 3.25 VideoWall Versión 3

Esta es la versión actual en la que se encuentra el *software* de video vigilancia que estoy desarrollando. Para esta versión me dieron la total libertad de organizar el proyecto como yo creía conveniente, por lo que decidí separar por completo la parte de video de la parte visual y crear con la solución de video una librería, de esta forma ya se tenía separado todo lo relacionado al tratamiento de video, de todo lo relacionado con la interfaz gráfica, con lo que se obtuvo un mejor ordenamiento del proyecto y también provocó que fuera más fácil de entender, documentar y resolver problemas.

Por otra parte me dedique durante algunos meses a resolver todos los problemas que tenía el video (de los cuales hablaré al final de la descripción de las mejoras que esta nueva versión tuvo), para después dedicarme a mejorar los aspectos visuales del *software*. En cuanto a la parte visual del *software* me pidieron que utilizara ***WPF*** en lugar de *Windows Forms* ya que con *WPF* se podía realizar animaciones más atractivas para el usuario, este cambio de alguna forma termino ayudándome ya que la personalización de los controles que formaban parte de la interfaz también fueron más fáciles de realizar, y el cambio entre un tipo de proyecto y otro no era demasiado por lo que prácticamente todo el código que había utilizado en *Windows Forms* me sirvió para el proyecto en *WPF*.

En el momento en el que se realizó este escrito, este proyecto contaba con 17 clases mientras que la versión 2 de este *software* contaba con 39 clases, como es evidente reduje a más de la mitad el número de clases, pero eso no fue todo ya que en la versión anterior casi todas las clases estaban formadas por alrededor de 1700 líneas de código cada una, muchas de las cuales eran líneas de código redundantes y sin propósito. Mediante una reorganización y uso de los diferentes conceptos referentes a la programación orientada a objetos se redujeron las líneas de código en la nueva versión a alrededor de 400 líneas de código por cada clase, aunque en algunas se redujo mucho más, pero eso será comentado más adelante.

Como se puede apreciar en la imagen de arriba el cliente seguía teniendo la misma organización ya descrita en la versión 2 de este *software*, por lo que en esta versión solo hablaré de las mejoras y agregados con los que cuenta. Uno de las principales y más notables funcionalidades que se le agregaron a esta versión, fue el control de usuarios, con el cual se lograba contar con un usuario administrador, el cual estaba pensado para un usuario de confianza ya que sería él quien se iba a encargar de agregar las cámaras y los usuarios a este *software*.

Menciono de confianza porque idealmente será el único que conocerá las contraseñas de los servicios utilizados para poder mover las cámaras y será él mismo quien pueda conceder permisos a otras personas para que puedan realizar estos movimientos. A los demás usuarios solo se les proporcionaba su usuario y contraseña para que pudieran hacer uso del *software*.

Para realizar esta administración de usuarios utilice una pequeña base de datos local la cual tiene el siguiente modelo.

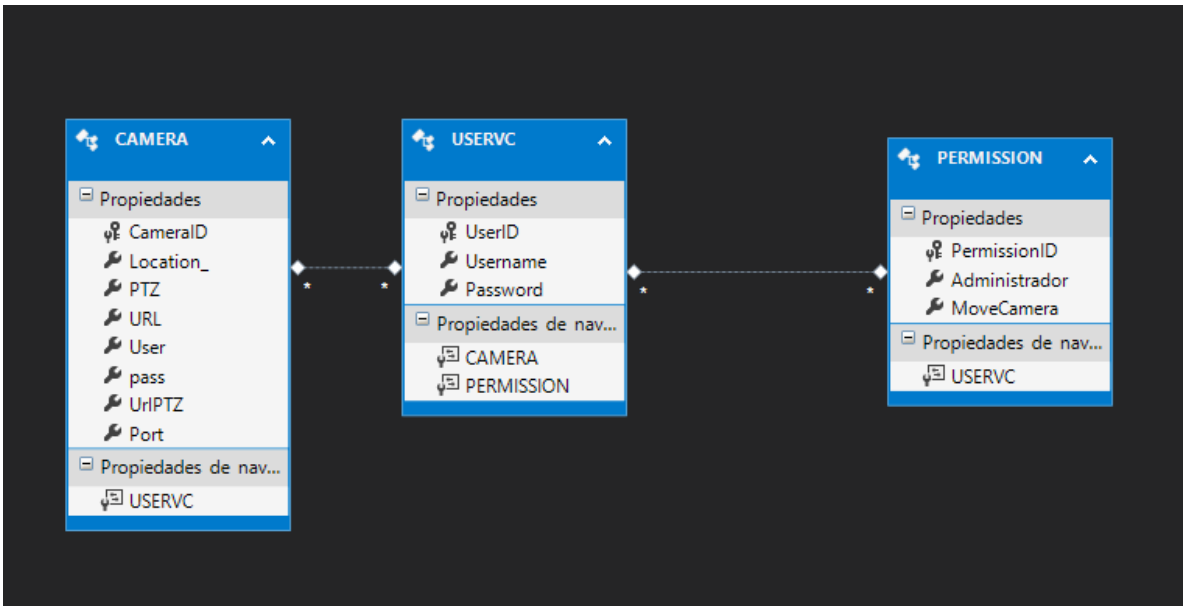


Imagen 3.26 Modelo de Base de Datos

Esta base de datos estaba pensada para poder agregar tanto usuarios como cámaras y poder asignarle a los usuarios tanto cámaras como ciertos permisos sobre esas cámaras. Para realizar las consultas desde *C#* hacia la base de datos utilice ***EntityFramework*** que por supuesto también es parte de .NET. Otra de las características de .NET que utilice para manipular la base de datos fue ***LINQ***, de este modo ya no tenía que tratar las tablas del modelo mostrado en la imagen anterior como una base de datos y hacer las consultas como tal, sino que *LINQ* combinado con expresiones lambda me dieron la facilidad de tratar a estas tablas como objetos, lo cual se me facilito un poco más.

Ahora bien la interfaz que interactuaba con esta base de datos para poder registrar nuevos usuarios, nuevas cámaras y sus relaciones fue llamada *UserCameraAdmin*. Cuando el programa era ejecutado por primera vez y al estar la base de datos vacía se le avisaba al usuario (imagen 3.27) que tenía que ingresar un nombre de usuario y contraseña y que este usuario sería tomado como administrador.

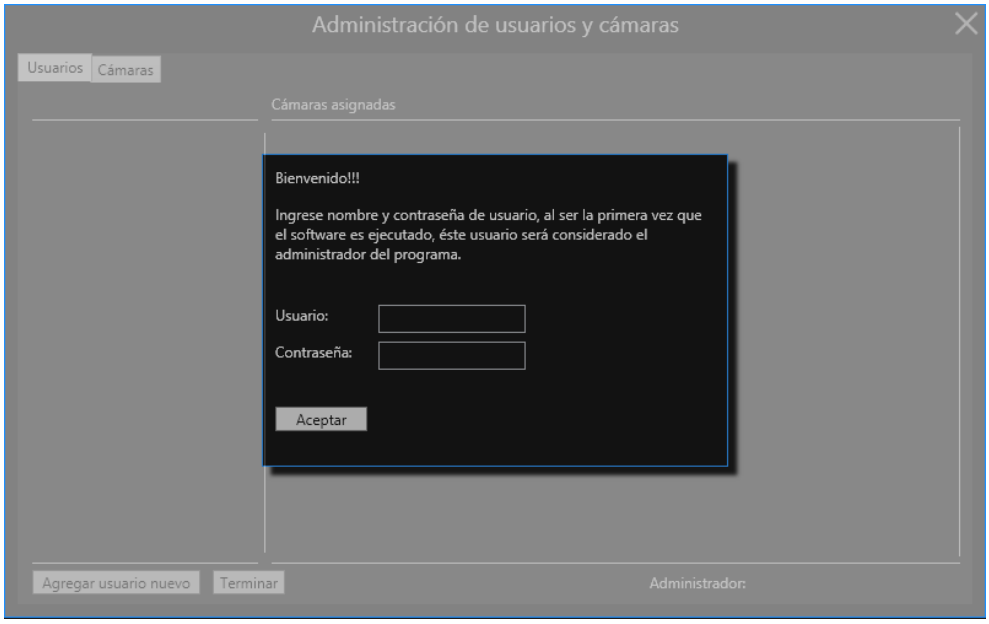


Imagen 3.27 UserCameraAdmin

Una vez ingresados los datos pedidos el usuario administrador contaba con dos pestañas diferentes:

- Pestaña Usuarios
- Pestaña Cámaras

La pestaña Usuarios se dividía en dos partes, la primera le mostraba al administrador la lista de usuarios que habían sido registrados en el programa (imagen 3.28) y si el usuario administrador seleccionaba a alguno de ellos podía ver las cámaras se le habían asignado y en ese mismo momento podía asignar más cámaras al usuario seleccionado, las cámaras que todavía no habían sido asignadas aparecían con opacidad y en el momento de ser asignada (presionando el botón “+”) la opacidad desaparecía.

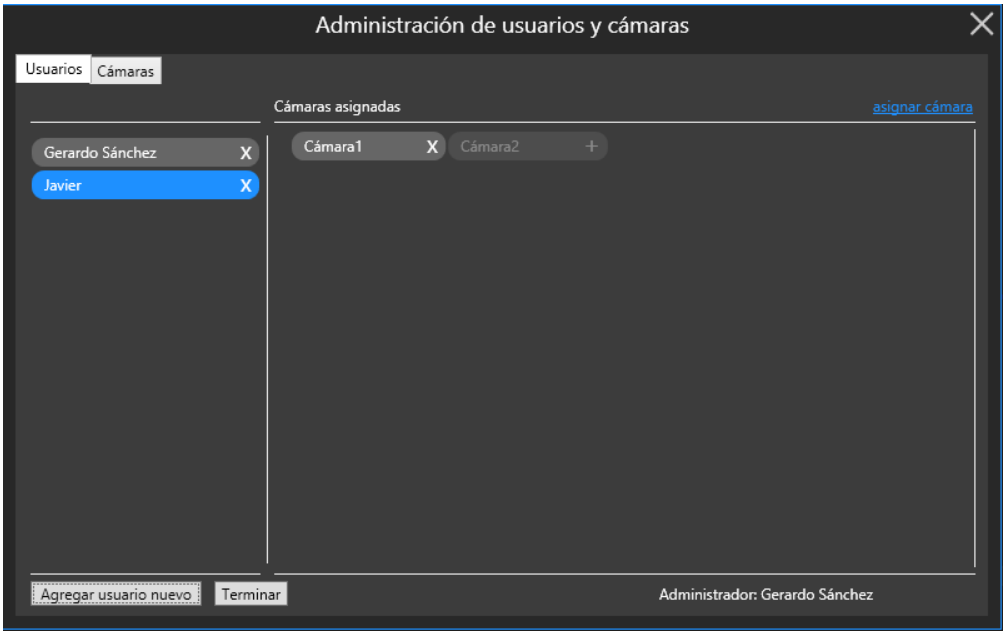


Imagen 3.28 Pestaña Usuarios

En ese mismo instante el administrador también podía quitar las cámaras al usuario que quisiera (presionando el botón “x”). En la interfaz mostrada arriba el administrador también tenía la posibilidad de borrar a los usuarios que quisiera. Para que el administrador pudiera asignar más cámaras a un usuario tenía que hacer clic en el texto que aparece de color azul en la parte superior derecha de la imagen mostrada arriba, en caso de que no hubiera ninguna cámara en la base de datos, se le mostraba al usuario un mensaje que le pedía que seleccionara la pestaña Cámaras para poder registrar cámaras y así posteriormente poder asignarlas.

La otra parte de la pestaña Usuarios aparecía cuando el administrador hacía clic en el botón “agregar usuario nuevo”, lo que se mostraba era un formulario para poder registrar a nuevos usuarios, asignándole su contraseña y nombre de usuario.

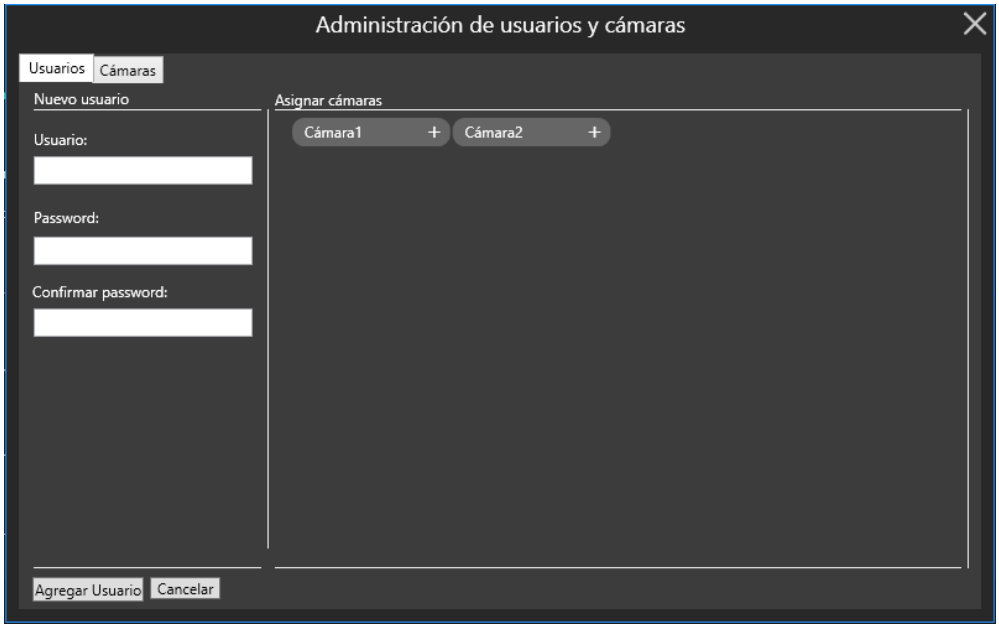


Imagen 3.29 Registro de usuarios

Un agregado a este formulario era que en el mismo momento en el que se estaba realizando el registro del usuario, se le podían asignar al mismo las cámaras que estuvieran disponibles en la base de datos, de esta forma el administrador podían ahorrar pasos en el registro y asignación de cámaras.

La pestaña Cámaras al igual que la pestaña Usuarios, estaba dividida en dos partes. La primera parte mostraba al usuario administrador las cámaras que ya habían sido agregadas a la base de datos (no tenían relación con ningún usuario) y se planeaba que también se mostrara información de la cámara como por ejemplo su ubicación, su id, la dirección a la que se tenía que conectar el *VideoDisplay* para empezar a transmitir, esto con el propósito de si llegara a haber un error en los datos, se pudieran corregir desde este panel (imagen 3.30).

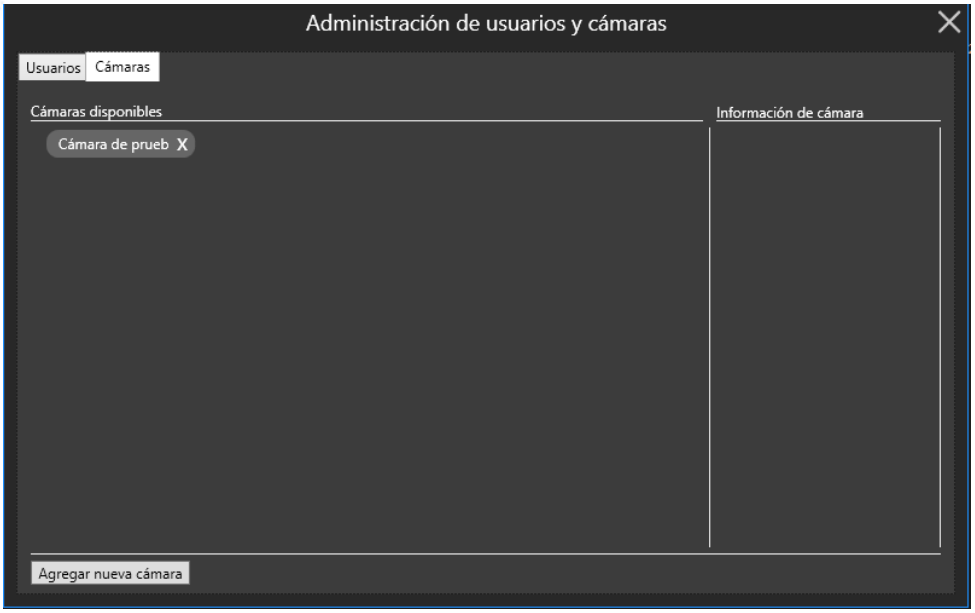


Imagen 3.30 Detalles de cámaras

Desde este panel también se podía realizar el eliminado de las cámaras haciendo clic en el botón “x”, lo cual provocaba que se borraran de la base de datos, y si es que existía alguna relación de estas cámaras con algún usuario dicha relación también era borrada. La segunda parte de la pestaña Cámaras (imagen 3.31) era en el que se realizaba el registro de las cámaras que iban a estar disponibles en el programa.

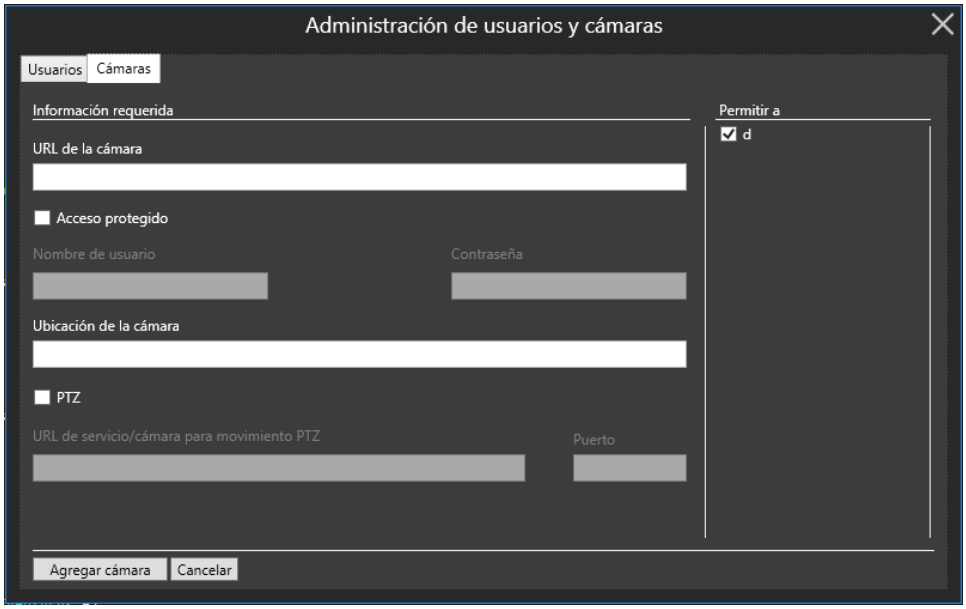


Imagen 3.31 Registro de cámaras

Los campos que eran necesarios para poder agregar la cámara, eran tanto la *url* de la cámara, como también el indicar la localización de la misma, adicionalmente se podía dar el caso de que alguna de las cámaras tuviera funcionalidades de *PTZ*, es decir que dicha cámara contara con las capacidades para poder ser movida, situación en la cual se podía marcar el acceso protegido, esta acción habilitaría los campos de nombre de usuario y contraseña los cuales eran necesario para hacer uso de un servicio *web* con el cual se podría enviar a la cámara los comandos para que hiciera los movimientos que el usuario requiriera.

Se podía dar el caso en el que la cámara *PTZ* no estuviera haciendo uso del servicio y se requiriera una conexión directa a la misma, razón por la cual se separaron el *CheckBox* de acceso protegido y el *CheckBox* de *PTZ*, ya que podía darse el caso que la cámara no requiriera contraseña y entonces solo se tenía que seleccionar que era *PTZ* e indicar la dirección a la que se iba a conectar y el puerto por que se iban a enviar las peticiones.

Adicionalmente en la parte derecha de esta vista se mostraba un listado de los usuarios que estaban registrados en el sistema y en ese mismo momento se le podía decir al sistema a que usuarios se le iba a permitir hacer uso de la cámara que estaba registrándose, el usuario administrador de forma predeterminada siempre aparecía marcado, por lo cual todas las cámaras iban a estar disponibles para él en caso de que quisiera visualizarlas.

Inicio de sesión

Una vez que se hubiera creado por lo menos el usuario administrador, se tenía la libertad de empezar a utilizar el programa, para poder saber qué usuario era el que estaba haciendo uso del programa y qué cámaras le corresponden a dicho usuario, se creó un *login* (imagen 3.32) mismo que al intentar hacer el inicio de sesión se conectaba a la base de datos para obtener las cámaras que le habían sido asignadas.

Imagen 3.32 Login

Si el usuario que inicio sesion en el sistema era el usuario administrador, la interfaz gráfica le mostrabaa un botón extra (imagen 3.33) con el que sería capaz de realizar la administración de usuarios y cámaras.

Imagen 3.33 Alta de usuarios

Si se trataba de un usuario común, simplemente se le dejaba ver las cámaras que tenía disponibles para visualizar y el botón de administración de usuarios y cámaras no aparecía. En caso de que fuera la primera vez que se utilizaba el programa o en caso de que el usuario hubiera decidido cerrar todas las cámaras que estaba visualizando antes de cerrar el programa, aparecía un panel del lado izquierdo de la interfaz gráfica que le mostraban las cámaras que tenía disponibles. En caso de que el usuario tuviera por lo menos una cámara en visualización este panel ya no aparecía a me nos que se presionara el botón que muestra dicho listado. Pero para continuar hablando de las nuevas funcionalidades del programa haré una descripción siguiendo los mismos puntos que se tocaron en el momento en el que se hizo la descripción de la segunda versión.

Menú superior

Visualmente constaba de los mismos elementos que en la versión anterior con la única diferencia de que se había agregado un nuevo botón en caso de que el usuario fuera el administrador del *software*. Lo que si cambio visualmente en esta versión fueron los dos primeros botones del menú.

Abrir configuración (Imagen 3.34) ahora presentaba una interfaz que concordaba con la interfaz del *software* e incluso era mucho más fácil de utilizar ya en esta ocasión el usuario no tenía que estar buscando sus archivos de configuración por el sistema de archivos de Windows, sino que todos estaba alojados en una misma ubicación y en la interfaz solo se le mostraba el listado de los que le pertenecían.

Imagen 3.34 Abrir configuración

El hecho de que el usuario solo tuviera que preocuparse por seleccionar el archivo desde el menú que se ve arriba, fue un gran cambio porque de este modo el usuario podía seleccionar la configuración que deseara mucho más rápido. Guardar configuración (imagen 3.35) también tuvo un cambio en su interfaz, el cual también mejoro el aspecto del *software*, pero además, permitió controlar en donde se guardaban los archivos de configuración ya que esta vez no permitía seleccionar una ubicación al azar sino que simplemente preguntaba con qué nombre se deseaba guardar la configuración y al aceptar simplemente lo guardaba.

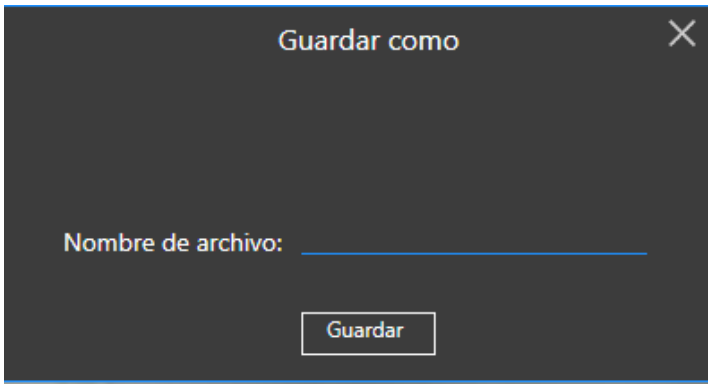


Imagen 3.35 Guardar configuración

Por supuesto ambos controles de usuario contaban con validación de los datos que se estaban proporcionando para que no se permitiera guardar archivos sin nombre o introducir caracteres especiales. En caso de que alguna de estas dos acciones ocurriera, el usuario era notificado por medio del cambio de color del control de usuario.

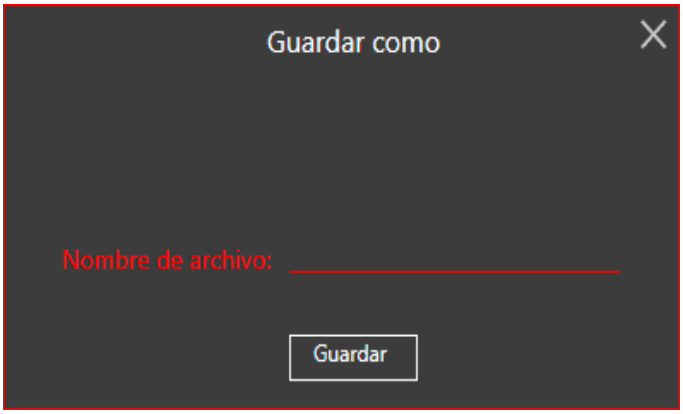


Imagen 3.36 Error al realizar guardado

De esta forma el usuario se veía obligado o a proporcionar un nombre para guardar la configuración de cámaras o a cancelar la acción de guardado de cámaras.

Una mejora interna que tuvo este *software* cuando se presionaba cualquiera de estos dos botones fue la serialización y deserialización del archivo de configuración ya que esta vez sí se hacía distinción entre una pestaña y las cámaras que le pertenecían a dicha pestaña, esto lo realice por medio de una serialización de objetos, el objeto serializado se trataba de un diccionario que como se sabe su estructura es <clave, valor> por lo que decidí que la clave fuera la pestaña en la que se encontraban las cámaras, mientras que el valor contenido en ese diccionario fuera una lista que se correspondía con las cámaras que estaban dentro de la pestaña cuando se decidió guardar el archivo de configuración. De este modo ya no se intuía que la cadena de texto que no se reconocía como cámara se trataba de una pestaña, sino que ya se sabía con precisión si era una cámara o una pestaña.

En caso de que la pestaña que el usuario intentaba guardar tuviera el mismo nombre que alguna otra de las pestañas, solamente hice lo mismo que el sistema de archivos de Windows, agregue un número extra al nombre de la pestaña, de este modo si existían tres pestañas llamadas por ejemplo “video” en el momento en el que se guardaba el archivo de configuración, estas pestañas pasaban a ser “video”, “video 1”, “video 2”, de este modo evitaba que existiera algún tipo de excepción debido a la utilización del diccionario con claves repetidas. El botón agregado de cámaras ahora no mostraba ese formulario en el que el usuario tenía que estar poniendo los datos de la cámara una y otra vez, ese trabajo era ahora realizado solamente por el usuario administrador, en esta versión el propósito de este botón cambio, ya que esta vez interactuaba con el listado de cámaras esta interacción lo que hacía era que por medio de una animación apareciera el listado de cámaras del panel lateral que se encontraba del lado izquierdo. De este modo el usuario ya solo tenía que agregar las cámaras sin necesidad de proporcionar ningún dato.

Una delas mejores actualizaciones que se le hizo a esta versión en comparación con la versión anterior fue la relacionada con las vistas, ya que como se había mencionado anteriormente, el código para cada vista se repetía demasiado además de que se tenían que estar haciendo demasiados cálculos para que el cambio entre cada vista funcionara.

Para esta versión decidí utilizar herencia ya que la funcionalidad de las vistas era la misma, solamente cambiaba la cantidad de cámaras que cada una de ellas podía soportar. El modelo quedo conformado por una vista padre llamada *ViewBase*, las diferentes vistas con las que contaba el *software* y una clase llamada *ViewManager*, gráficamente se podría resumir de este modo.

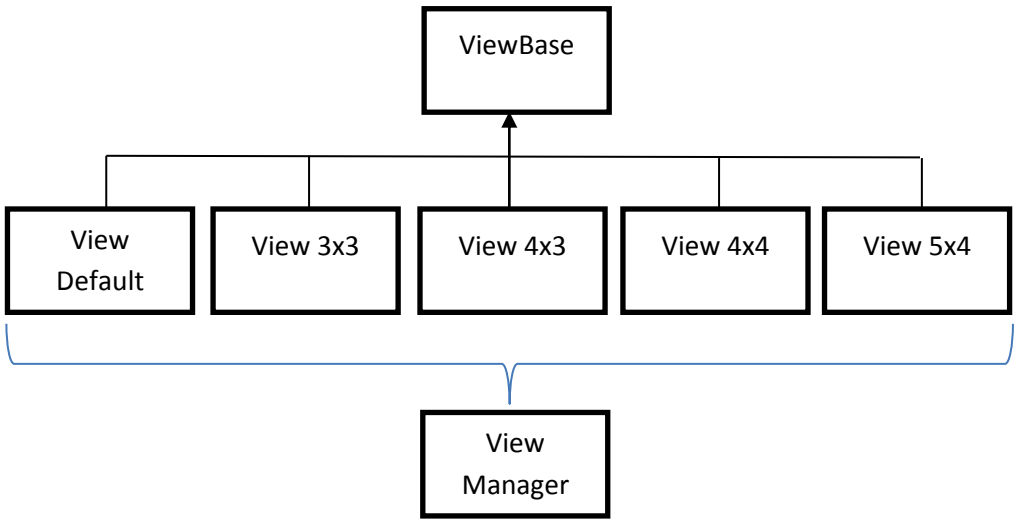


Imagen 3.37 Modelo de herencia de Vistas

La clase *ViewBase*, como su nombre lo indica era la clase base de las vistas, en ella se encontraba prácticamente todo el código utilizado por todas las vistas para que estas pudieran interactuar con el usuario. Era en esta clase en la que ahora se llevaba a cabo la funcionalidad del arrastre de las cámaras, también se utilizaba esta clase para realizar el agregado de cámaras, la eliminación de cámaras y el *fullscreen*. Uno de los campos o propiedades más importantes con los que contaba la clase base era la propiedad *Capacity* mediante la cual, cada una de las vistas le indicaba a la clase padre la cantidad de cámaras que podía soportar, y en base a esa cantidad de cámaras era como la clase base manejaba el límite de creación de cámaras, con el propósito de no desperdiciar memoria creando cámaras que ya no se iban a poder visualizar. Otro atributo importante que tenía la clase base era una referencia a los contenedores donde se iban a posicionar las cámaras en cada vista, por medio de estas referencias la clase base solamente agregaba *VideoDisplays* a la vista y estas se posicionaban en automático.

Las vistas en este caso tuvieron un gran cambio ya que ahora la parte lógica de las mismas solo se dedicaba a enviarle información a la clase base en el momento en el que se creaba un objeto de la vista, toda la funcionalidad era llevada a cabo por la clase base. La parte visual de las vistas estaba conformada por dos capas (imagen 3.38).

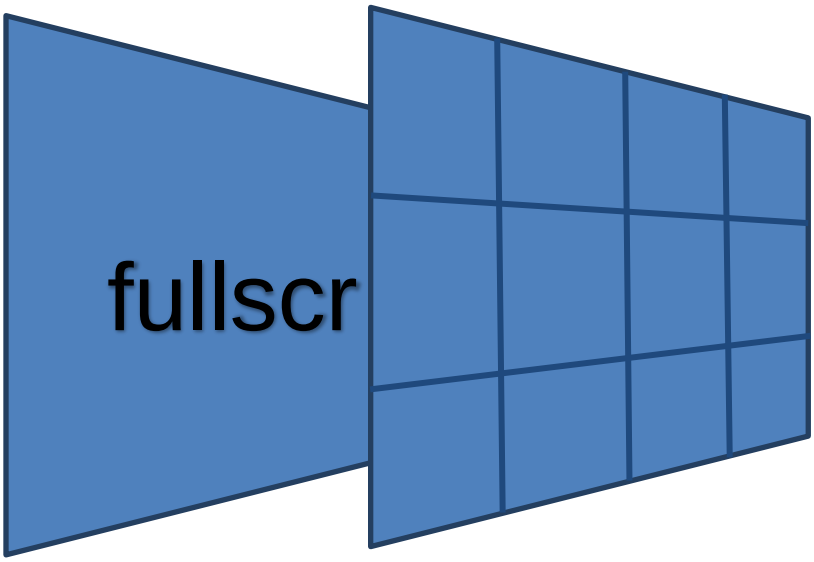


Imagen 3.38 Estructura de Vistas

La capa principal era con la que se topaba el usuario final una vez que se hubiera creado su usuario, esta capa era un *grid* en el cual se iban colocando cada uno de los *VideoDisplay* que el usuario tenía permitido, esto hasta que se alcanzaba la capacidad máxima de la vista que en ese momento hubiera estado seleccionada. Para evitar el tener que estar manejando posiciones en el *grid* por renglón y columna, decidí agregar un *DockPanel* en cada uno de los elementos que conformaban dicho *grid*, la referencia a estos elementos era la que se enviaba a la clase padre para que pudiera manipular los *VideoDisplay* que tenía la vista. La otra capa era otro *DockPanel* que tenía el tamaño de todo el *grid*.

Por medio de este *DockPanel* es como se podía realizar el *fullscreen*, esto se lograba removiendo el *VideoDisplay* del *DockPanel* que se encontraba en el *grid* y agregándolo al *DockPanel* que se encontraba detrás del *grid*, en este caso se tenían que realizar dos pasos en lugar de uno como se realizaba en la segunda versión, ya que en la versión anterior simplemente se tenía que agregar el video al panel que se encontraba detrás del *grid*, a diferencia de esta última versión, en la que primero se tenía que remover el video del *grid* para después agregarlo al *DockPanel* ya que de lo contrario el software mostraba una excepción indicando que el *VideoDisplay* ya estaba agregado a una colección.

En el *VideoDisplay* se agregó un botón más en su interfaz de usuario y este aparecía solo si el usuario administrador marcaba a una cámara como ***PTZ*****,** este botón aparecía tanto si la cámara visualizada estaba en *fullscreen* como también aparecía si no estaba en *fullscreen*, cuando la cámara no estaba ocupando toda la pantalla, el botón desencadenaba el evento de *fullscreen* y también el evento que permitía que el usuario pudiera mover la cámara a su antojo, para esto, haciendo uso de un mensaje mostrado en la pantalla durante algunos segundos, se le hacía saber al usuario que podía mover la cámara simplemente haciendo clic sobre la imagen en la dirección en la que deseaba mover la cámara. Ahora bien si la cámara estaba en *fullscreen* y se pulsaba este cuarto botón, se llevaba a cabo el efecto contrario, es decir, se regresaba la cámara a tamaño normal y se quitaba la suscripción al evento de movimiento. El hecho de hacer y dejar de hacer *fullscreen* a la cámara fue pensando en el hecho de que en ese momento el usuario deseaba enfocarse en la cámara que iba mover y realizar este procedimiento permitía que el usuario centrara su atención en una sola cámara.

Una de las principales diferencias o mejor dicho una de las diferencias que existía entre *WPF* y *Windows Forms* era la forma en la que se mostraban las imágenes que procedían del flujo que enviaba la cámara ya que para poder mostrar el vídeo en *Forms* se utilizaba un control llamado *PictureBox*, mismo que aceptaba un objeto tipo *Image*, mientras que *WPF* utilizaba un objeto de tipo *ImageSource* por lo que para poder visualizar las cámaras en esta nueva tecnología tuve que agregar un poco de código extra, mismo que se encargaba de pasar el flujo recibido, al formato que utilizaba el *ImageSource*.

Ahora bien existe una clase más que se encargaba de administrar el cambio de vista, esta era *ViewsManager* que lo que hacía era cambiar la vista dependiendo de la selección que el usuario realizaba en el menú superior, además de cambiar la vista, también cambiaba las referencias de los *VideoDisplay* para que aparecieran en esta nueva vista. Hablando de optimización, al utilizar herencia cada una de las vistas estaba conformada por 40 líneas de código, mientras que la clase base contaba con 270 líneas de código, para lograr esta enorme reducción de líneas de código, intente cuidar cada detalle que conllevaba la interacción entre el usuario y las vistas.

En la versión anterior, cuando el usuario pasaba el mouse sobre alguno de los *VideoDisplay*, estos cambiaban su borde de color con el propósito de llamar la atención del usuario y que se enfocara en dicho *VideoDisplay*, pero esta funcionalidad estaba basada en la vista, lo cual provocaba que todo el procedimiento de saber si el mouse estaba o no dentro de algún *VideoDisplay* se llevaba a cabo por medio de la consulta de las coordenadas del *mouse* y de su interacción con el *VideoDisplay*, la consulta de las coordenadas se realizaba por si el *mouse* estaba dentro del *VideoDisplay* pero sobre alguno de los cuatro botones que formaban parte del *VideoDisplay* (*Remove*, *fullscreen*, *Snapshot* y *PTZ*), situación en la que el *software* pensaba que ya estaba fuera de este control y el borde se quitaba, mediante dichas coordenadas se podía saber si el *mouse* aún se encontraba dentro del *VideoDisplay*, pero aun así había ocasiones en las que el borde continuaba visualizándose aunque el mouse ya no estuviera dentro del *VideoDisplay*. Gracias a *WPF*, en esta tercera versión del *software* esta funcionalidad fue mucho más rápida de realizar y más eficiente, ya que solamente tuve que utilizar un componente de ***XAML*** llamado ***Triggers*** e indicarle al *software* que en caso de que el *mouse* estuviera dentro del borde del *VideoDisplay*, tenía que cambiar el borde de color y en caso contrario no tuve que hacer nada debido a que *WPF* sabía que ya no tenía que aplicar este cambio y el *VideoDisplay* regresaba a su estado normal.

Otro aspecto que también se solucionó y que también fue mucho más fácil de aplicar en *WPF* fue el arrastre de las cámaras ya que como se había mencionado en la versión anterior, el aspecto visual que se tenía en el momento en el que se intentaba realizar el arrastre de las cámaras no era muy adecuado, en esta nueva versión, cuando se iniciaba el arrastre de una cámara, el *mouse* se posicionaba en el centro del video que se estaba arrastrando (Imagen 3.39) de esta forma, la sensación de estar arrastrando dicho control era mucho más real.

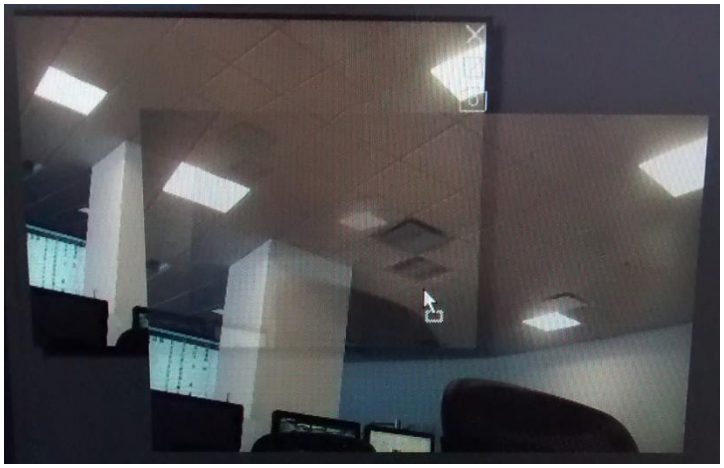


Imagen 3.39 Arrastre de VideoDisplay

Listado de cámaras

El listado de las cámaras mostrado o mejor dicho descrito en la versión anterior era visualmente pobre por lo que en esta versión se mejoró su aspecto visual para que fuera más agradable al usuario. El aspecto de cada una de las cámaras que el usuario administrador había agregado al cliente de video era la siguiente.

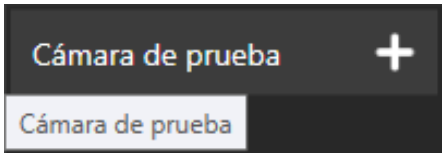


Imagen 3.40 Elemento de listado de cámaras

Como es posible apreciar, en éste cada una de las cámaras que eran listadas estaban formadas por el nombre de la cámara, que en el caso de la imagen corresponde a “Cámara de prueba”, un control de .Net llamado *ToolTip*, el cual era utilizado para mostrar el nombre completo de la cámara sobre la que estuviera el puntero del *mouse*, así en caso de que el nombre que se encontrará en el listado fuera demasiado grande, el usuario del *software* sería capaz de ver su nombre completo y por último se tenía el símbolo “+” el cual hacía que, además de mejorarlo visualmente, también tuviera funcionalidad misma que consistía en poder agregar la cámara a la vista que se tenía seleccionada en ese momento, ya que sin este nuevo botón en el listado, no habría forma de que el usuario pudiera agregar las cámaras a la vista para empezar el monitoreo de las mismas.

Ahora bien el listado de las cámaras ahora estaba dividido en dos categorías, en la primera (categoría de cámaras disponibles) se mostraban todas aquellas cámaras que habían sido asignadas al usuario y que no estaban siendo monitoreadas, mientras que en la otra categoría (categoría de cámaras en reproducción) se mostraban todas aquellas cámaras que el usuario ya había agregado a la vista para que éstas fueran monitoreadas. Cuando el usuario hacía clic en alguno de los signos "+" la cámara a la que referenciaba automáticamente empezaba a cargar y el elemento del listado que se encontraba en "cámaras disponibles", pasaba ahora a ocupar un lugar en el listado "de cámaras en reproducción" para estos dos controles se utilizaba el mismo control de usuario, solo que cuando la descripción de la cámara pasaba al listado "cámaras en reproducción", el botón con el cual empezaba a reproducirse, cambiaba su visibilidad a oculto.

Imagen 3.41 Cámaras disponibles y cámaras en reproducción

Las funcionalidades de la versión anterior en la que al pasar el mouse por el listado se iba seleccionando en la vista la cámara a la que le correspondía, seguía conservándose ya que esa era una forma rápida de ubicar alguna cámara de interés. También como se puede observar en la imagen anterior, en la categoría de cámaras disponibles había una equis cuya funcionalidad es ocultar el listado de cámaras mismo que se mostraba haciendo clic en el botón "cámaras" descrito anteriormente. Cuando se mostraba este listado de cámaras, ocurría una animación que mejoraba visualmente el aspecto del *software*, además de que la redimensión de todos los elementos que conformaban la interfaz gráfica era automática lo que también ahorro líneas de código.

Visualización de cámaras

En cuanto a la visualización de cámaras en esta ocasión, el comportamiento variaba un poco. Al ser *WPF* mucho más amigable en cuanto a la creación de interfaces en comparación con *Windows Forms*, esta vez no tuve que sobre escribir ningún control, sino que solo tuve que aplicar un estilo en el cual le agregué a mí *TabControl* una pestaña con un pequeño símbolo "+" para que fueran agregadas las demás pestañas y a esta misma pestaña le quite la funcionalidad de poder ser seleccionada ya que su única función era la de permitir insertar una nueva pestaña al *TabControl*. En este estilo también configure el *TabControl* para que agregara las pestañas en el orden que yo necesitaba, esto es, cada que se agregaba una pestaña yo quería que apareciera al lado izquierdo de la pestaña que contaba con la funcionalidad de agregar más pestañas. Para esto en el momento de la creación solo tenía que cambiar el índice de la nueva pestaña.

Al utilizar estilos en lugar de código duro evite la construcción de una clase llamada *CustomTabControl* ya que todo la funcionalidad que esta clase me proporcionaba, la pude realizar por medio de los estilos aplicados. Con esto logré ahorrar aún más memoria además de que le evite al *software* la sobre escritura del control para que lograra serme de provecho. Ahora bien por defecto cada vez que el usuario arrancaba el programa se agregaba al *TabControl* de .Net una nueva pestaña y en caso de que el usuario ya hubiera iniciado sesión y dejado algunas cámaras en la sesión anterior se evitaba la construcción de la pestaña por defecto para dar paso a la carga de las pestañas que se tenían en la sesión anterior.

Como consecuencia cada que se agregaba una pestaña, también se tenía que construir y agregar un nuevo manejador de vistas, mismo que tenía todos los atributos inicializados para que el usuario pudiera agregar más cámaras. En cuanto a la funcionalidad que se tenía anteriormente del cambio de nombre de la pestaña, creé un control de usuario (Imagen 3.42) que era mostrado cuando el usuario realizaba un doble clic sobre alguna de las pestañas, con esto era más claro para el usuario que la acción de cambiar el nombre estaba siendo desencadenada, ya que en la versión anterior el cambio de nombre era sobre la misma pestaña.

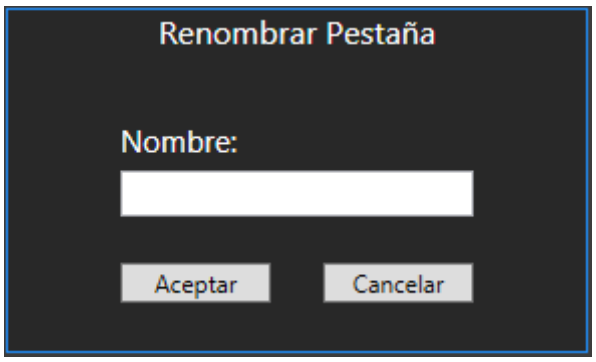


Imagen 3.42 Renombrar pestaña

El control para el cambio de nombre no era cerrado hasta que el usuario proporcionaba un nombre o hasta que hiciera clic en el botón cancelar, con lo que la pestaña conservaba el nombre que tenía antes de que se hubiera deseado cambiar su nombre, el hecho de cerrarse solo en estas dos situaciones fue con el propósito de evitar que las pestañas tuvieran el nombre vació.

Ahora bien, a cada una de las nuevas pestañas que aparecía en *TabControl* le agregué un botón con el símbolo “x”, cuando el usuario hacía clic sobre este botón el *TabControl* se comunicaba con el manejador de vistas de la pestaña y este manejador de vistas se comunicaba con la vista que actualmente estaba seleccionada en esa pestaña para indicarle que debía entrar en un ciclo para que empezara a eliminar cada una de las cámaras que estaban dentro de la vista y así liberar la memoria del programa. Como ya se había mencionado en la versión anterior, las pestañas fueron utilizadas para que el usuario pudiera visualizar mayor cantidad de cámaras, hasta el momento el *software* solo soportaba alrededor de 60 cámaras, cuando se llegaba a esa cantidad de cámaras el programa alcanzaba el GB de memoria *RAM* y en ese momento Windows cerraba el programa (Imagen 3.43) notificando que había ocurrido un error e internamente generando una excepción en la librería **mscorlib.dll** de Microsoft.

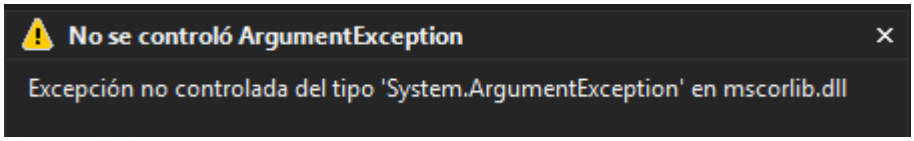


Imagen 3.43 Excepción en mscorlib.dll

En esta versión se analizó el problema de la cantidad de cámaras soportadas y se generaron dos opciones parar poder soportar muchas más cámaras.

La primera fue inspirada en el navegador *Chrome*, ya que este navegador lo que hace es crear un proceso por cada una de las pestañas que se están utilizando en el buscador (imagen 3.44), lo que le permite que cada una de esas pestañas tenga su propia memoria asignada y por ende si estas capacidades fueran aplicadas al cliente de video, cada una de las pestañas con la vista *default* activada alcanzaría un máximo de 200 a 240 MB por pestaña, mientras que la aplicación que desencadenaba todos estos procesos pasaría de ocupar 1GB a solamente alrededor de 40MB.

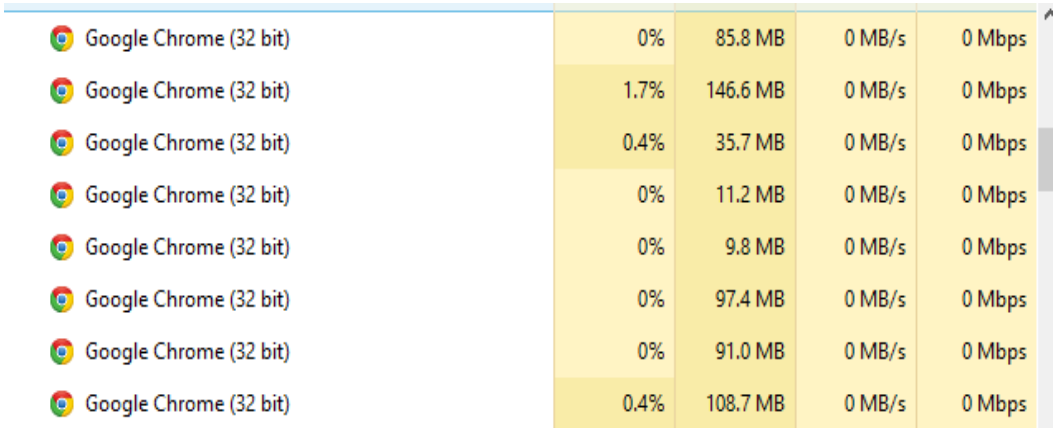


Imagen 3.44 Procesos Chrome

Esta opción sería una buena solución de no ser por el hecho de que el usuario solo podía visualizar una sola pestaña a la vez por lo que la memoria de los demás procesos (pestañas) estaría siendo desperdiciada, pero la ventaja de esta primera opción era que cuando se hiciera un cambio de pestaña, el usuario vería de inmediato el video de las cámaras. Pero si por ejemplo tuviéramos alrededor de 11 pestañas en nuestro programa, cada una con 20 cámaras, el usuario solo estaría sacándole utilidad a 200-240MB de los más de 2GB que el programa estaría demandando, esto para el usuario, dependiendo de las características de su equipo podría convertirse en un consumo excesivo de los recursos de su sistema. Como alternativa a esta opción y de hecho fue la que se utilizó al final, fue simplemente, sabiendo que como se iba a tener solo una pestaña activa a la vez solo esa pestaña sería la que iba a estar consumiendo memoria mientras que las cámaras que estaban en las demás pestañas solo iban a ser almacenadas como datos para que en el momento en el que se volviera a activar una pestaña, se tomarían estos datos y simplemente las cámaras de dicha pestaña volvieran a ser iniciadas.

En este caso la mayor ventaja sería en si para el consumo de recursos ya que el *software* completo sin importar la cantidad de cámaras que tuviera solo estaría consumiendo de 200 a 240 MB pero a consecuencia de esto cuando el usuario decidiera cambiar de pestaña tendría que esperar algunos segundos para que las cámaras volvieran a ser inicializadas ya que como se mencionó, solo estarían guardadas como datos.

Antes de empezar a hablar de la librería de vídeo que desarrolle solo falta mencionar que algunas de las funcionalidades del cliente que fueron retiradas debido a que pasaron a ser obsoletas, no estaban completas como para ser incluidas o simplemente no tenía sentido que fueran incluidas ya que solo significaban mayor cantidad de líneas de código sin alguna funcionalidad provechosa, estas funcionalidades fueron el *EditSource* que aparecía en el *VideoDisplay* ya que los usuarios comunes ya no iban a poder modificar ningún dato de las cámara, ya solo estaban limitados a utilizarlas, el botón de reconexión, ya que este procesos ahora se llevaba a cabo automáticamente, la reproducción de vídeos grabados se quitó debido a que solo se podían reproducir hasta el momento vídeos almacenados de manera local y esta funcionalidad no estaba planeada para eso y por último en las vistas también se decidió quitar el arrastre de cámaras en la que la cámara de mayor tamaño podía moverse de un lado a otro ya que no servía para nada que esto se pudiera realizar.

Una vez comentadas las funcionalidades de esta versión pasaré a hablar de la librería que cree para todas las cuestiones relacionadas con la codificación y decodificación del video y de cómo fueron enfrentados los problemas mencionados en la versión anterior.

Como ya se había mencionado estos problemas eran principalmente tres:

- Visualización de dos vídeos en un solo *VideoDisplay*.
- El aparentemente no refrescado del *VideoDisplay* (fantasmas).
- Repetición de *frames*.

Cabe mencionar que en un principio esta librería de vídeo llamada en esta última versión *VideoCoreLibraries* en un momento llegó a ser al 100% basada en los proyectos de ejemplo que proporcionaba *streamcoders*, razón por la cual se presentaban estos problemas.

Para poder resolverlos, tuve que ayudarme del depurador de *Visual Studio* con el cual fui siguiendo la ejecución del programa. Gracias a este depurador pude dar solución al primero de los problemas mencionados, ya que pude notar que en la versión dos de este cliente se estaba haciendo un mal manejo de los hilos ya que los hilos que utilizaban los *VideoDisplay* para poder visualizar el video, estaban cruzándose o mejor dicho, todos los *VideoDisplay* llegaban a un mismo método, el cual era el que se utilizaba para mostrar el video, por lo que ocurría una condición de carrera y cualquiera de los *VideoDisplay* que llegaba a este método podía cambiar la referencia de en donde se iba a mostrar dicho video así es que lo que pasaba era que un *VideoDisplay* llegaba inmediatamente después del otro, cambiaba la referencia y hacia que el que estuviera delante de él mostrara el video en donde no le correspondía. Para solucionarlo simplemente bloquee el código en el que se mostraba el video hasta que el hilo que llegaba primero terminara de realizar todo el procedimiento necesario para mostrar el video.

En cuanto al refrescado del video, este defecto provocaba que se viera como una especie de fantasma en el video y además mientras iban pasando los *frames*, dicho video se veía arrastrado, ya que estaba tratando de actualizarse, fue resuelto cuando me di cuenta que todo estaba relacionado a un *frame* llamado *keyframe* el cual era importante para identificar el cambio de *frame* y era el que estaba causando los problemas de los fantasmas y a pesar de que este *frame* era indispensable para saber cuándo ocurría dicho cambio, no lo era tanto para las cuestiones visuales por lo que realice algunas pruebas evadiendo estos *keyframes* al momento que iban a ser mostrados y las pruebas funcionaron, ya que en el video no volvió más a verse con este efecto.

El último problema que se presentaba en el video, el relacionado con la repetición de *frames*, fue más un problema de codificación ya que por error cree dos hilos que estaban referenciando al mismo método, este método se encargaba de mostrar el flujo de video. Para solucionar el problema tan solo era necesario quitar uno de los hilos, en cuanto cancele dicho hilo todo funciono mejor, pero al final se tomó la decisión de que esta parte del *software* se llevara a cabo mediante eventos ya que de este modo solo se ejecutaría el código en caso de que existieran *frames*, mientras que con el uso de hilos se tenía que estar monitoreando si existía un nuevo *frame* para que en ese momento se mostrara.

En cuanto a la organización de esta librería, se utilizaron solamente cuatro clases para poder dar soporte a las peticiones que el usuario podía realizar, estas clases fueron:

- *MediaSource*
- *RtspConfiguration*
- *RtspSession*
- *SubstreamDescriptor*

MediaSource se encargaba principalmente de iniciar la reproducción del flujo de video que era proporcionado como parámetro mediante un objeto de tipo *RtspConfiguration* del cual hablaré más adelante, en *MediaSource* se llevaba a cabo la decodificación de dicho flujo de video, una vez que éste flujo de video era decodificado, se desencadenaba uno de dos eventos, estos eran llamados *onNewFrame* y *onNewFrameBmpSource* y esto era dependiendo de si se utilizaba *WinowsForms* o *WPF* para crear la solución de *software* en la que se iba a visualizar el video ya que en un principio el *software* solo estaba desarrollado en *WindowsForms* y por cuestiones del cambio de tecnología fue que cree el segundo evento, estos eventos eran precisamente los encargados de la visualización del video y para esto se enviaba el flujo de video haciendo un *cast* a tipo *Image* para que el manejador de eventos del objeto que se suscribía a uno de estos eventos se encargara de manipular el *frame* recibido.

Otra de las acciones que podía realizar esta clase era detener la reproducción del flujo de video, notificaba la pérdida de conexión con la cámara así como también llevaba a cabo el reinicio de la reproducción del flujo de video en caso de que este se perdiera. Cada una de estas acciones contaba con su propio método asignado, por ejemplo, para iniciar la conexión a alguna cámara se utilizaba el método *Play* en el cual se intentaba levantar una sesión con la cámara que el usuario requería y en caso de que dicha sesión se lograra de forma satisfactoria, en este mismo método una instancia de la clase *SubStreamDescriptor* era creada y se suscribía al evento que decodificaba el video para que el usuario final pudiera ser capaz de empezar el monitoreo de la cámara en cuestión.

El manejador de eventos que se encargaba de dicha decodificación era llamado *DecoderProcessor* y una vez que realizaba esa decodificación enviaba el *frame* resultante al método *PlayoutPicture* quien recibía dicho *frame* como parámetro y después de esto desencadenaba el evento para que el manejador de eventos que se encontraba en la clase *VideoDisplay* ocurriera, este manejador de eventos tenía dos tratamientos diferentes dependiendo de si se utilizaban los formularios de Windows o si se utilizaba *WPF* porque como ya se explicó anteriormente, la forma en que ambas tecnologías mostraban el video era diferente.

Es prácticamente evidente que como esta clase tenía la capacidad de reproducir o reconectarse a un flujo de video, también debería de ser capaz de detener la reproducción de dicho flujo de video. Para lograr detener la reproducción se utilizaba el método *Teardown* en el cual le ponía fin a la sesión que mantenían el cliente y la cámara y después se liberaba la memoria. Había ocasiones en las que de un momento a otro, se perdía la conexión con la cámara o cámaras que el usuario final estaba visualizando y gracias a que *streamcoders* detectaba cuando esto ocurría, fue que cree el evento *Session_OnDisconnect* en el cual como la desconexión a la cámara era involuntaria, decidí realizar la reconexión a la misma, para lo cual lo primero que tenía que hacer era terminar la sesión de la cámara conservando los datos de la misma para así poder realizar nuevamente la configuración y conexión a dicha cámara, esto ocurría así ya que no había forma de recuperar la sesión actual para volverla a activar, este procedimiento de terminar la sesión e iniciar una nueva se llevaba a cabo en el método *ResetSession*.

Ahora bien, el método *TearDown* antes mencionado se encargaba de parar el flujo de video, pero este método se encontraba encapsulado para que no pudieran hacer uso de este de manera directa, para que una instancia que utilizara esta clase pudiera detener el flujo de video desde otro lugar que no fuera la propia clase, tenía que utilizar el método *StopPlaying* el cual sí era público y por ende cualquiera que hiciera uso de esta clase podría verlo y tener acceso a él para su ejecución.

La clase *RtspConfiguration* era mucho más sencilla que la antes mencionada, la instancia de esta clase solamente se encargaba de almacenar datos de la cámara, almacenaba información tal como la *URL* de la cámara a la cual se iba a conectar, en caso de que esta tuviera usuario y contraseña, también era capaz de almacenar esa información, si se querían almacenar datos más específicos de la conexión se podía también indicar la resolución que se prefería, para que en el momento en el que se estableciera la sesión, se tomara este tipo de parámetros en cuenta para la recepción de los *frames*

Además de esta información esta clase solo manejaba un método en el cual lo único que se hacía era verificar la *URL* que el usuario había proporcionado y separaba los distintos campos que esta tenía, de esta manera la clase era capaz de almacenar la *IP*, el puerto, el método de entrega que iba a utilizar así se podían tener detalles más concretos de la cámara en caso de que en algún momento fueran a necesitarse.

La clase *RtspSession* es la que tenía como tarea administrar todas las acciones relacionadas con la sesión que era mantenida ya sea entre el cliente de video y entre la cámara o entre el cliente de video y el servidor de video. Para realizar dicha administración esta clase utilizaba a la clase *RtspConfiguration* para obtener de ella toda la información que necesitaba para levantar dicha sesión. También hacía uso de la clase *SubstreamDescriptor* la cual permitía validar el tipo de códec de video que se deseaba consumir, para de este modo elegir el decodificador adecuado para dicho flujo. Esta es solo una de las tareas que podía realizar la clase *SubstreamDescriptor*, más adelante describiré con mayor profundidad sus capacidades.

Ahora bien en el momento en el que se creaba una instancia de esta clase, era cuando se almacenaba información tal como la dirección *URL* a la que el usuario deseaba conectarse, en caso de que dicha cámara o en si la conexión necesitara acceder por medio de un usuario y una contraseña también estos campos eran almacenados, esto era gracias a que el constructor recibía como parámetro esta información, misma que al ser utilizada a lo largo de la clase, fue almacenada de manera global para que todos tuvieran acceso a ella dentro de la clase.

Para poder saber si el flujo de video del que quería disponer el usuario estaba disponible, se utilizaba un manejador de sesiones el cual hacía uso de la información antes mencionada para encontrar el flujo de video y en caso de que este flujo estuviera disponible para su consumo, simplemente se devolvía un objeto que indicaba dicha situación, en caso de que no fuera así se devolvía *null* y se liberaba la memoria que era utilizada por el manejador de sesiones. En este método también se podía especificar la cantidad de tiempo que se quería esperar por una respuesta pero al final decidí no establecer un límite ya que la conexión a las cámaras que utilizaban la tecnología *3G* era demasiado tardada y si se especificaba un tiempo límite era casi imposible que pudiera establecerse la sesión con dicha cámara y por ende el usuario no podría ver el video.

En el momento en el que se desarrolló esta tercera versión el cliente de video solo soportaba dos tipos de códec, estos eran H.264 y **MPEG-4**, entonces esta clase contaba con un método llamado *ParseSession* que era capaz de verificar si el flujo de video demandado utilizaba alguno de los dos códec antes mencionados y de ser así la sección del flujo de video que era analizada era marcada como útil, esta verificación de las diferentes secciones que componían al flujo se hacía continuamente hasta que la sesión terminara, para que el usuario de este método supiera si el códec era útil se le devolvía un objeto de la clase *SubstreamDescriptor* y en caso de que el códec de video no fuera soportado, se devolvía *null*, de este modo el usuario de esta clase podía saber si debía continuar con el procesamiento del video o si debería ignorarlo.

En caso de que alguna de las acciones antes mencionadas no funcionara, simplemente se hacía uso del método *Teardown* mismo que se encargaba de liberar la memoria consumida por el manejador de sesiones y también liberaba la memoria consumida por la instancia de la clase *discoveryReport*, como este método podía ser llamado en cualquier instante e inclusive por diferentes hilos, decidí bloquear la llegada de dicho hilo hasta que completara todas las actividades que se tenían que realizar para el hilo en cuestión ya que en algún punto intente realizar el proceso de liberación de memoria y los hilos modificaban la información que compartían y entonces se generaba una excepción.

La última clase que compone a esta solución era la clase llamada *SubstreamDescriptor*, esta clase fue realmente complicada analizarla ya que era la clase que mayormente utiliza los componentes de *streamcoders* y como ya había mencionado anteriormente, *streancoders* no proporcionaba una buena documentación por lo que era difícil comprender lo que se realizaba en esta clase, afortunadamente al estar tanto tiempo interactuando con ésta, he podido comprender si bien no todo, algunas de las funcionalidades que realizaba esta clase. En general era en esta clase en la que se creaban los hilos que eran necesarios para procesar los *frames* provenientes del flujo de video, así como también se creaba el hilo que se encargaba de desencadenar el evento para que el objeto de la clase *MediaSource* llevara a cabo el proceso de decodificación de video.

Esta clase también hacia la diferenciación entre los frames que eran procesados por completo y haciendo uso de diferentes *buffers* era como guardaba estos *frames*, así como también la información que contenían dichos *frames*, otro de estos *buffers* también era utilizado para guardar los paquetes *RTP* y así como era capaz de realizar estas acciones, también es evidente que debería de tener algún modo de terminar con todo este procesamiento, por lo que la clase también contaba con los métodos necesarios para liberar la memoria.

La clase, como cualquier otra iniciaba con su constructor, el cual no tenía alguna actividad en específico, solamente se encargaba de inicializar los *buffers* que eran utilizados a lo largo del programa y también le indicaba al usuario de la misma que en ese momento el funcionamiento de esta clase estaba activo. Una vez inicializados los atributos, el usuario de la clase, que en este caso era una instancia de la clase *RtspSession* tenía que hacer uso del método *Parse* en la cual por medio de un *switch* se verificaba el tipo de códec que se estaba utilizando y en base a eso se elegía el decodificador adecuado para poder realizar el proceso de decodificación, al momento de ser elegido, este decodificador también era iniciado y se agregaba como parte de la configuración del video.

Como ya se mencionó la clase *MediaSource* era la encargada de realizar la decodificación del video y esto era gracias al uso del método *processFrame* en el cual se verificaba que existieran frames completos y en caso de que así fuera, se generaba un objeto *VideoFrame* obteniendo datos de dicho *frame* tales como su alto, su ancho, etc. Después de esto el *VideoFrame* era agregado a uno de los *buffers* utilizados en esta clase. Esta actividad se llevaba a cabo mientras la sesión con la cámara siguiera existiendo.

Otro de los métodos con los que contaba esta clase y que también era utilizado por la clase *MediaSource* era el método *Attach* el cual dependiendo del método de entrega (*DeliveryMethod*) utilizado era como se configuraba la sesión, este método también servía como inicializador de los hilos que llevaban a cabo el procesamiento del flujo de video y la decodificación del mismo, es decir era en este método en el que se suscribían los eventos a su manejador de eventos. El método contrario a esta clase era el método *Dettach*, en el que se le indicaba a los usuarios de esta clase que el funcionamiento de la misma estaba desactivado para que por su cuenta liberaran los recursos que dichos usuarios consumían, en este método también se detenían los hilos que habían sido iniciados en el método *Attach* y por último se limpiaban los *buffers* que tuvieran información, para que la memoria utilizada por esta clase pasara a estar completamente disponible para su utilización.

En la descripción del método *Attach* mencioné que se creaban hilos mismos que eran suscritos a manejadores de eventos, uno de esos manejadores de eventos era el método *sub_OnInterleaveData*. Este manejador de eventos era utilizado para actualizar un reporte de envío, mismo que contenía la información de la calidad de los paquetes que iban llegando desde la cámara a la que se conectaba el cliente, así como también dicho reporte contenía información del ritmo con el cual se iban actualizando esos paquetes. Cada uno de los paquetes ***RTP*** que llegaban eran recibidos haciendo uso del método *process_ReceivePacket* y su tarea era verificar si dicho paquete estaba bien sincronizado, en caso de que no fuera así el paquete era rechazado y en caso de que estuviera bien sincronizado dicho paquete era agregado a un *buffer* del cual se extraía cuando era decodificado.

Por último, el otro manejador de eventos que era utilizado era el *videoDecoderThreadProc* el cual empezaba a funcionar hasta que una cierta cantidad de *frames* completos estuviera almacenada en el *buffer* de *frames* completos, una vez esto se cumplía, se verificaba la información contenida esos en *frames* y la agregaba al *buffer* que contenía esa información. Para que los *buffers* no se llenaran demasiado y por ende la memoria *RAM* que ocupaba este *software* no fuera creciendo, en el momento en el que se procesaba cada uno de los *frames*, estos se iban extrayendo del *buffer* en el que se encontraban así como también la información que estos contenían era liberada del otro *buffer*.

Bugs de la versión 3

Una vez descritos las dos soluciones junto con sus respectivas clases y porque no todo en el cliente estaba correcto al cien por ciento, ahora mencionare cuales eran los *bugs* que se me presentaron en esta tercer versión y que en versiones posteriores a esta, se les ira buscando la solución más adecuado posible. Estos *bugs* ocurrían tanto en la interfaz gráfica como en la lógica aplicada al *software*. Para las cuestiones relacionadas a la lógica, en realidad hasta el momento en el que se escribió este documento, en la solución *VideoCore* solamente se había presentado un problema el cual todavía no era identificado de manera concreta, solamente se sabía que ocurría cuando las cámaras que estaban conectadas al servidor dejaban de funcionar todas al mismo tiempo, esto podía ocurrir por ejemplo en caso de que la corriente eléctrica que consumían las cámaras dejara de fluir. El motivo por el cual no se pudo detectar a que se debía el problema generado por esta desconexión simultánea con las cámaras fue debido a que en ese momento el *software* estaba en periodo de pruebas y no en modo de depuración, con lo cual solo se me mostró un mensaje indicándome que el *software*, por alguna razón desconocida había dejado de funcionar.

Esta situación hasta el momento en el que fue realizado este escrito no había ocurrido de nuevo, pero con el propósito de que este *bug* se pudiera capturar si es que llegaba a ocurrir nuevamente, todos los días al terminar mi jornada laboral dejaba funcionando el *software* en modo de depuración, de este modo podría darme cuenta de lo que lo provocaba y así podría darle solución.

En cuanto a la solución *VideoClient* los *bugs* eran más fáciles de resolver ya que estaban bien identificados. Uno de estos *bugs* ocurría cuando se quería eliminar un *VideoDisplay* de alguna de las vistas, ya que estos, como ya se había mencionado estaban en una colección pero se accedía a ellos por medio de su nombre, por lo que si ocurría el caso que por accidente o por decisión propia del usuario administrador uno o más *VideoDisplay* tuviera el mismo nombre que algún otro, se generaba una excepción ya que la **expresión lambda** que se encargaba de eliminar dicho *VideoDisplay* se confundía porque encontraba más de una coincidencia y no sabía cuál de estas eliminar. Pero como ya mencione al tener identificado el problema solo era cuestión de cambiar su comportamiento para que funcionara correctamente solo que para cuando se realizó este escrito aún no estaba corregido este error.

Otro de los errores que se presentaba, y que más que un error era una cuestión estética, era el hecho de que algunos de los controles todavía no estaban personalizados en cuanto al aspecto visual se refería, por lo que durante la descripción de este escrito, en algunas de las imágenes mostradas, se podía notar como por ejemplo los botones que formaban parte de la interfaz que utilizaba el usuario para interactuar con la base de datos (Imagen 3.27) aún se veían como los controles de usuario que vienen por default en *Visual Studio* para crear interfaces gráficas, esto estaba así porque principalmente quería enfocarme en la funcionalidad del programa para que posteriormente pudiera hacerme cargo de la estética del mismo.

También cabe mencionar que el proyecto no estaba terminado porque aunque ya cumpliera con los requerimientos que se me habían dado, todavía podía hacerle mejoras al proyecto o agregar nuevas características que permitirían que el usuario fuera capaz de optimizar sus tiempos al momento de estar utilizando el programa, estas características eran tanto propuestas mías como requerimientos extras que en una versión posterior tendría que agregar.

Capítulo 4. Resultados

Resultados del proyecto

Considerando como lucía este *software* desde que inicio el proyecto a la evolución que ha tenido hasta su última versión es posible darse cuenta que aunque todavía se le tienen que agregar más funcionalidades y pulir algunas de las que se han desarrollado, el proyecto ha ido evolucionando de forma satisfactoria.

Uno de los aspectos más notables de esta evolución fue en la que considero la pieza más importante del proyecto, ya que es la que le da sentido, porque al tratarse de un proyecto de vídeo vigilancia se tenía que procurar que la visualización de vídeo fuera lo más clara posible y gracias a que dedique un periodo de unos meses en estar analizando todos los problemas que se presentaban en el vídeo, logré que el procesamiento de vídeo que se realizaba y los problemas que dicho procesamiento causaba fueran corregidos.

Tan grande fue la mejora en el rendimiento que pasó de tener solo cuarto cámaras en acción simultánea a alcanzar alrededor de sesenta, y aunque otras empresas que tienen proyectos similares ofrecen la visualización de una mayor cantidad de cámaras, al estar analizándolas me di cuenta que al pasar la visualización de un grupo de cámaras a otro, en realidad lo que hacían era volver a cargar las cámaras de ese otro grupo como si se hubieran agregado nuevamente al cliente de vídeo, cosa que con el cliente que yo desarrolle no ocurría ya que las sesenta cámaras siempre estaban activas y el usuario no tenía que esperar a que las cámaras cargaran para que pudiera verlas. Solo que al final se decidió aplicar la misma técnica que las otras empresas con el propósito de ofrecer al cliente la carga de una mayor cantidad de cámaras.

Otro de los aspectos a destacar en mi punto de vista fue en cuanto a la intuitividad del *software*, ya que a la mayoría de los usuarios se les quitó la responsabilidad de estar llenando formularios de configuración para establecer una relación entre el *software* y las diversas cámaras que este usaría, de esta forma se logra que la productividad de quien monitoriza aumente y que por supuesto sólo se preocupe por la única tarea que debería hacer, monitorear.

Hablando de la mejora de la productividad del usuario, otro aspecto que estoy seguro que también la mejora es el hecho de que puede crear su propio grupo de cámaras y guardar esa configuración en un archivo de configuración mismo que le permitirá en cualquier momento simplemente seleccionar dicho archivo, cargarlo en el *software* y sin tener que hacer nada más, empezar a realizar su trabajo, o mejor aún en caso de que el usuario siempre utilice la misma configuración de cámaras, él ya no tendrá siquiera que preocuparse por elegir algún archivo de configuración ya que este se guarda y se abrirá automáticamente en cada inicio de sesión. Esto mejorará aún más los tiempos que tardarán los usuarios antes de poder empezar a monitorear.

En general estos son los resultados más importantes y los que pueden ofrecer mayores beneficios a quienes hagan uso del *software*, ya sea que se trate de alguna tienda departamental o del gobierno de algún estado.

Resultados personales

Personalmente los proyectos que he realizado en la empresa me han permitido tanto aplicar mis conocimientos como conocer cuáles son las capacidades con las que cuento para poder obtener más conocimientos y por supuesto saber aplicarlos de la mejor forma posible. Destaco por ejemplo el hecho de que cuando la empresa me cambio de proyecto, yo no había utilizado el lenguaje C#, pero aun así acepte el cambio y comencé a aprenderlo y que cuando se me asignó el proyecto de vídeo vigilancia este era un completo desastre y hoy en día es el más estable de los clientes de vídeo que se ha desarrollado en la empresa.

Esto lo logre gracias a que mi lógica esta cada día mejor desarrollada y por supuesto a que pude darle un orden a las diferentes partes que lo formaban separando la interfaz del *software* de la lógica del mismo y así pude atacar cada problema que tenía el proyecto de forma aislada, simplemente aplique lo que un profesor de la universidad nos decía "divide y vencerás" y fue así como fui puliendo cada detalle del *software* y aunque puede ser posible que alguien con mayor experiencia podría mejorar el orden que yo le di, quedo satisfecho con el trabajo que he realizado hasta el momento porque sé que poco a poco aprenderé más y que las próximas versiones de este *software* serán aún mejores.

Conclusión

Tomando en cuenta el avance actual del proyecto de video vigilancia y de acuerdo con los objetivos comentados en la descripción del proyecto principal, puedo decir que la mayoría de los objetivos han sido cumplidos, algunos de ellos todavía no se integran al programa pero ya están funcionando como módulos independientes pero en próximas versiones serán incluidas como parte del *software*. Cabe mencionar que no han sido incluidas hasta el momento debido a que mi principal objetivo era que los fallos que ocurrían en el video fueran corregidos en la medida de lo posible y que el hecho de simplemente agregar más características al *software*, provocaría que dichos errores fueran creciendo y que incluso fueran mucho más difíciles de corregir.

Hablando ahora del cumplimiento de los objetivos comenzaré hablando del primero de ellos y por supuesto del más importante, la visualización del video. Tras varios meses trabajando con la librería de video que construí y como lo comente anteriormente, los errores hasta el momento que se desarrolló este escrito fueron corregidos, a excepción de uno de ellos que no había detectado el hecho del porque ocurría, pero que se continua intentando encontrar la razón que causa este error.

El control de cámaras con movimiento *PTZ* también se cumplió y se logró con una manera de uso sencilla, en la que solo se tiene que hacer clic sobre la zona a la que se desea enfocar la cámara y eso bastara para mandarle un comando a la cámara que hará que se mueva. Tal vez algo que le falta al control de cámaras *PTZ* es el *zoom* es decir la parte Z de la palabra y no se agregó por el simple hecho de que los drones que desarrolla la empresa todavía no pueden hacer *zoom*. Pero esta característica del control de cámaras estaba planeada para realizarse con el *scroll* del *mouse* de manera que cuando se haga *scroll* hacia adelante la cámara se acercara al objetivo y cuando el *scroll* se mueva hacia atrás, la cámara se alejara de su objetivo. Y se planeó así porque las capacidades del *zoom* iban a ser agregadas a los drones posteriormente.

La grabación de video es una característica que se incluyó en la parte del servidor del proyecto de video vigilancia, por lo que por el momento, por *default* toda cámara a la que se conecte el cliente de video y que por supuesto se encuentre en el servidor, en automático se realizará la grabación de su trasmisión, por el contrario, las cámaras a las que el cliente se conecte de forma directa, no tendrán esta capacidad de grabación, por lo que solo servirán para visualización de video.

Como se había comentado en este escrito la capacidad de poder visualizar los videos grabados solo se había logrado de manera local, por lo que no se incluyó en el proyecto de video vigilancia ya que dichos videos estaban planeados para ser proporcionados por el servidor.

Para la tercera versión del *software* descrito, ya se podía visualizar el video que se encontraba almacenado en el servidor, pero todavía faltaba agregarle algunas características que todos los reproductores de video tienen, estas características eran por ejemplo, la capacidad de poder pausar el video en el momento requerido, detener el video o regresar a un punto anterior del mismo, es decir, el cliente podía demandarle al servidor que le proporcionara un video pero simplemente estaba limitado a verlo hasta que dicha grabación terminara. Es por esa razón que tampoco en la tercera versión del cliente de video, se permitió realizar la visualización de videos grabados.

Como mencione en el inicio de la descripción del proyecto de *software*, los diversos programas que son realizados en Global Corporation son modulares, por lo que el módulo de detección de movimiento ya lo había realizado en el proyecto de contador de personas en el que explique que ese contador también podía se enfocado a esa actividad, por ende tan solo con agregar este módulo al cliente de video, se estaría cumpliendo con esta característica, misma que ya ha sido probada.

En la tercera versión de este *software* se introdujo la capacidad de poder tener un usuario administrador, mismo que se iba a encargar tanto de crear tantos usuarios como necesitara como de agregar las cámaras que cada usuario podía utilizar y los permisos que dichos usuarios podían tener sobre las cámaras, con esto se cumplía con el control de usuarios de la aplicación, así como también la asignación de los niveles de usuario que existen en el cliente de video.

Glosario

3G.

Se refiere a la tercera generación de la telefonía móvil, esta generación envuelve a sus predecesores. La Unión Internacional de Telecomunicaciones la definió para facilitar el crecimiento y el incremento del ancho de banda así como para soportar mayor cantidad de aplicaciones ya que tiene una mejor eficiencia espectral. Dicho espectro va desde los 400 Mhz a los 3GHz.

.NET

Es una plataforma de desarrollo para compilar aplicaciones de Windows, *Windows Phone*, *Windows Server* y *Microsoft Azure*. Está formado por *Common Language Runtime* (CLR) y la biblioteca de clases de .NET que incluye clases, interfaces y tipos de valor que son compatibles con una amplia gama de tecnologías. .Net *Framework* proporciona un entorno de ejecución administrado, un desarrollo e implementación simplificados e integración con una gran variedad de lenguajes de programación incluidos *VisualBasic* y *C#*.

Praxair

Es una empresa proveedora de gases industriales fundada en 1907.

Texas Instruments

Es una empresa norteamericana que desarrolla y comercializa semiconductores y tecnología para ordenadores, destaca por ser el mayor suministrador de circuitos integrados para móviles.

Digi-Key

Es el cuarto más grande distribuidor de componentes electrónicos en Norte América y el quinto más grande a nivel mundial.

Newark Element14

En un distribuidor de componentes electrónicos que trabaja en Norte América y partes del centro y Sudamérica. Fue fundada en 1934 como Newark Electronic. Hoy en día es el sexto distribuidor más grande a nivel mundial.

Linux

Su nombre correcto es GNU/Liunx y se trata de un sistema operativo de software libre basado en UNIX. Su base es el núcleo llamado Linux (Kernel Linux) desarrollado originalmente por Linus B. Torvalds a principio de la década de los noventa.

Leopard Imaging Inc.

Es una compañía localizada en Fremont, California, enfocada principalmente a las cámaras *Web* de alta definición, sistemas de video conferencia, cámaras *IP* de seguridad, cámaras estereotípicas 3D y módulos de cámaras móviles.

Kernel

Es el corazón del sistema operativo de GNU/Linux y es el encargado de que el *Software* y el *Hardware* de una computadora con este sistema puedan trabajar en conjunto. Sus principales funciones son: la administración de la memoria para programas en ejecución, la administración del tiempo del procesador y la administración de la interacción con los periféricos de una computadora.

gstreamer.

Es un Framework multimedia libre y multiplataforma fundado en el año de 1999 por Erik Walthinsen. Escrito en el lenguaje C, usando la librería *GObject*. Este Framework permite crear aplicaciones audiovisuales como de video, sonido o codificación.

FPS

Son las siglas de *Frame Per Second* o cuadros por segundo en español y se trata de la medida de la frecuencia a la cual un reproductor de imágenes reproduce distintos cuadros (imágenes).

PTZ

Hace referencia a *Pan-Tilt-Zoom* y es la capacidad que tienen algunas cámaras de poder realizar movimientos y acercamientos de forma remota, dichos movimientos los puede realizar de forma horizontal o vertical.

API

Son las siglas de *Application Programming* Interface o interfaz de programación de aplicaciones y Es el conjunto de funciones, métodos o procedimientos que una librería de *software* puede ofrecer, esto con el propósito de que otro *software* pueda hacer uso de estos métodos y el desarrollador no tenga que volver a escribir su funcionalidad.

SDK

Son las siglas de *Software Development Kit* o *kit* de desarrollo de *software* y se trata de un conjunto de herramientas de desarrollo de *software* que facilitan al programador la creación de aplicaciones para un sistema en específico.

OpenCV

Es una librería de código abierto para el desarrollo de aplicaciones de visión por computadora, esta liberado bajo la licencia BSD con lo que se permite hacer libre uso de ella.

EmguCV

Es un envoltorio multiplataforma de la librería de procesamiento de imágenes *OpenCV*, el cual permite que lenguajes de programación .NET puedan ser capaces de llamar a los métodos o funciones de OpenCV.

Framework

Es una estructura conceptual que proporciona módulos de *software* concretos tales como programas, bibliotecas, lenguajes, los cuales ayudan a desarrollar y unir diferentes componentes de un proyecto.

Aforge .NET

Es una librería de visión por computadora e inteligencia artificial la cual fue desarrollada por Andrew Kirillov para el framework .NET.

H.264

Es una tecnología de compresión de video desarrollada por la *ITU* (International Telecommunication Union pos sus siglas en inglés) () y la *ISO (International Organization for Standarization*)/*IEC* (*International Electrotechnical Commission*). En su arquitectura H.264 está definido en las especificaciones de MPEG-4 el cual está conformado por 28 especificaciones, de las cuales las siguientes cuatro son las más relevantes para el streaming:

- Parte 2-MPEG video
- Parte 3-MPEG-4 audio
- Parte 10-Advanced Video Coding
- Parte 14 Container Format

Los objetivos de este códec de video eran principalmente: incrementar la eficiencia en la compresión, brindar soporte para aplicaciones de video especiales tales como videoconferencias, Almacenamiento *DVD*, emisión de video, y *streaming* sobre internet. Otro de sus propósitos era incrementar la fiabilidad de la transmisión.

RTSP

Es un protocolo de nivel de aplicación para controlar la entrega de datos con propiedades de tiempo real como audio y video, este protocolo está destinado a controlar múltiples sesiones de entrega de datos proporcionando un medio para elegir la entrega de dichos datos tal como *UDP* (*User Datagram Protocol*), *UDP* multicast y *TCP* (*Transmission Control Protocol*). También proporciona un medio para elegir los mecanismos de entrega basados en RTP (*Real-time Transport Protocol*).

Soluciones Visual Studio

Las soluciones contienen los elementos necesarios para crear una aplicación en *Visual Studio*. Una solución incluye uno o más proyectos, archivos y metadatos que ayudan a definir la solución en conjunto. *Visual Studio* genera automáticamente una solución al crear un proyecto.

Eventos

Es el modo que tiene una clase para proporcionar notificaciones a los clientes de la clase cuando ocurre algo de reseñar en un objeto.

WPF

Es un sistema de presentación de próxima generación para crear aplicaciones cliente de Windows que proporcionen una experiencia de usuario impactante

Entity Framework

Es un asignador objeto-relacional que permite a los desarroladores de .NET trabajar con datos relacionales usando objetos específicos del dominio. Elimina la necesidad de la mayor parte del código de acceso a datos que los desarrolladores suelen tener que escribir.

LINQ (Language Integrated Query)

Conjunto de características presentado en *Visual Studio 2008* que agrega capacidades de consulta eficaces a la sintaxis de los lenguajes *C#* y *Visual Basic*. *LINQ* incorpora patrones fáciles y estándar para consultar y actualizar datos.

XAML

Es el acrónimo de eXtensible Application Markup Language y se trata del lenguaje utilizado para instanciar objetos de .NET, su rol principal es construir interfaces de usuario para *WPF*.

Triggers

Es una forma declarativa mediante la cual se pueden automatizar cambios de estilos simples sin la necesidad de escribir código en el lenguaje *C#*

MPEG-4

Es un estándar de datos audiovisuales, surgió en Mayo de 1991 y que no solo se preocupaba por la compresión de audio y video, sino que también se dedicó a crear un conjunto de herramientas (*toolbox*) para negociar con productos audiovisuales en general las cuales estaban basadas en objetos. Otra de sus características era la interoperabilidad que le permitía intercambiar cualquier tipo de dato, sea texto, gráficos, video o audio. Este estándar está orientado a cumplir ocho puntos que considera importante en relación con el manejo de audio y video y que fueron formulados en una reunión realizada por un comité llamado como el estándar “MPEG-4”, estos puntos son los siguientes:

1. Herramientas de acceso multimedia basados en contenido. El estándar debe proporcionar herramientas para acceder y organizar datos.
2. Edición de manipulación y flujos de bits basados en contenido. El estándar debe tener una sintaxis y un esquema de codificación.
3. Codificación de datos híbridos naturales y sintéticos. Una escena natural es producida por una cámara de video. Una escena sintética consiste de texto y gráficos.
4. Mejora del acceso aleatorio temporal. Los usuarios pueden querer acceder a ciertas partes del archivo comprimido, el estándar debe incluir etiquetas que faciliten el acceso a dichas partes del archivo.
5. Mejorar la eficiencia de codificación. Mejorar compresión.
6. Codificación de múltiples flujos de datos concurrentes. Incluir interacción con las imágenes del flujo audiovisual.
7. Robustez en entornos propensos a errores. El estándar debe proporcionar código para corrección de errores para transmisión a través de canales ruidosos.

Escalabilidad basada en contenido. El flujo comprimido puede incluir datos en buena resolución y alta calidad.

RTP

Es un protocolo de red para entregar audio y video sobre redes *IP*, es utilizado extensivamente en sistemas de comunicación y entretenimiento.

Expresión lambda

Es una función anónima que se puede usar para crear tipos delegados. Utilizando expresiones lambda puede escribir funciones locales que se pueden pasar como argumentos o devolverse como valor de llamadas a funciones. Fueron introducidas en la versión 3.0 de Framework de .NET.

Unicast

Envío de información desde un único emisor a un único receptor. En el caso del video el flujo es transmitido al origen de la petición *rtsp* y el puerto es elegido por el cliente.

Multicast

Envío de información desde múltiples emisores hacia múltiples receptores de forma simultánea. En este caso el servidor puede elegir la dirección y el puerto o el cliente puede elegir la dirección a la que se va a conectar esto se da por ejemplo cuando se participa en una conferencia.

DeliveryMethod

Son los métodos de entrega utilizados por *streamcoders* para transmitir el flujo de video, en el desarrollo de este escrito el cliente de visualización de video siempre utilizo *rtspinteleave* como método de entrega.

Referencias

David Salomon, et al, (2007), “Data Compression The Complete Reference”, (4a ed.), Computer Science Department California State University Northrige, CA, Springer-Verlag London Limited.

Hector Facundo Arena, (2003), “La Biblia de Linux”, Primera Edición, Ciudad de Buenos Aires, Argentina, MP Ediciones.

Jan Ozer, (2013), “Producing Streaming Video for Multiple Screen Delivery”, Primera Edición, 412 West Stuart Drive Galax VA 2433, Doceo Publishing Inc.

Matthew MacDoald, (2012), “Pro WPF in C# 2010 Windows Presentation Foundation in .NET 4”, (4a ed.), Aprees.

Network Working Group, (Abril de 1998), Real Time Streaming Protocol, http://www.ietf.org/rfc/rfc2326.txt.

Microsoft Developer Network, Visual C# (Consultas varias relacionadas con programación en C#), http://msdn.microsoft.com/

Bernát Gabor, How to build applications with OpenCV inside the Microsoft Visual Studio,http://docs.opencv.org/doc/tutorials/introduction/windows_visual_studio_Opencv/windows_visual_studio_Opencv.html#windows-visual-studio-how-to.

EmguCV, EmguCV Tutorial, http://www.emgu.com/wiki/index.php/Tutorial

Mahvish Nasir, EmguCV with C# (Tutoriales),http://fewtutorials.bravesites.com/tutorials

Anexos

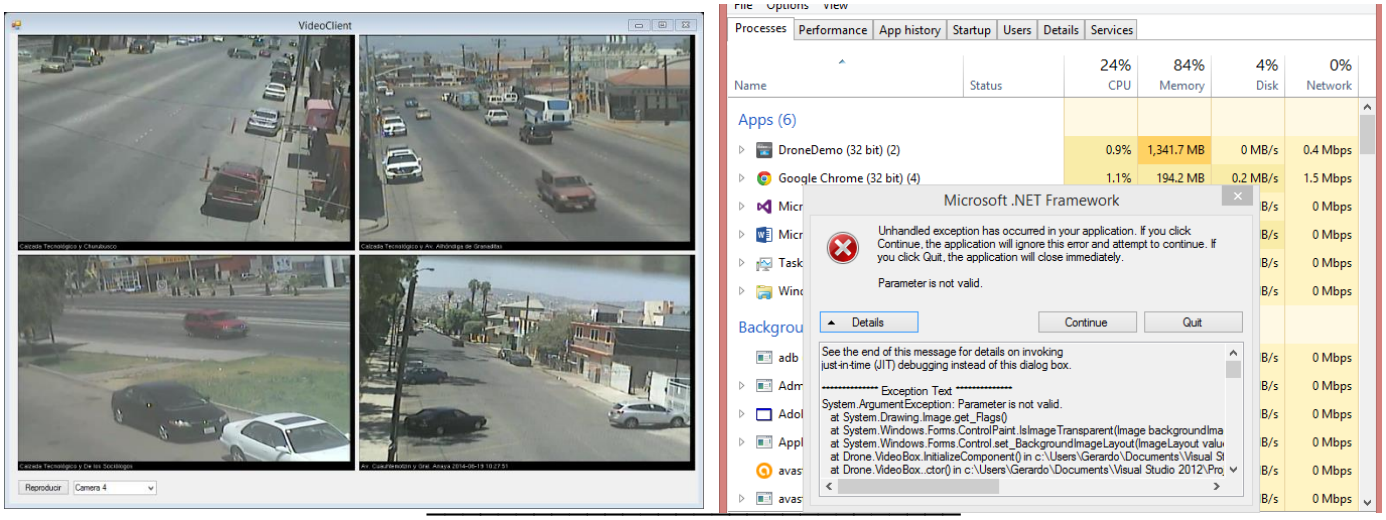
Pruebas de estrés

Durante las tres versiones que hasta el momento había tenido el proyecto de video vigilancia se llevó a cabo un registro de las diferentes pruebas que se le iban aplicando a este *software*, estas pruebas eran realizadas agregando cámaras al cliente de video hasta que este no soportara más y terminara generando errores, esto con el propósito de saber cuántas cámaras se iban a poder reproducir de forma simultánea y como era que dicha cantidad iba afectando el rendimiento del sistema operativo y de la aplicación en sí.

A continuación iré mostrando algunas de las pruebas realizadas al *software* en sus diferentes versiones, así como también mostraré capturas del administrador de tareas en el que se puede visualizar la cantidad de memoria *RAM* ocupada por el *software*. La versión uno como ya se había explicado era estática y solo alcanzaba a soportar cuatro cámaras por lo que fue ese el límite que pudo soportar para probarlo.

Prueba con cuatro cámaras primera versión.

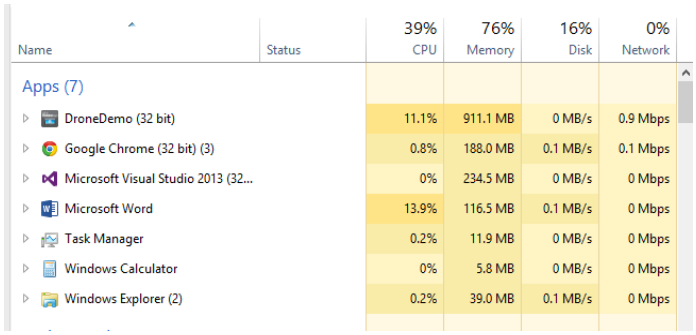
Esta fue una de las primeras pruebas que realice en la cual se puede observar que, con tan solo cuatro cámaras en funcionamiento, el programa superaba el *Gigabyte* de memoria y también se puede observar que solo consumía esa cantidad debido a que se generaba la excepción mostrada, de no ser así la memoria hubiera seguido creciendo.



Anexo 1.0 Prueba primera versión

Para la segunda versión, se verifico primero la cantidad de memoria que el software demandaba por el solo hecho de ser ejecutada, y para mi sorpresa dicha cantidad de memoria era demasiada, alrededor de 230 MB por el solo hecho de abrir el programa, todavía sin cargar al *software* ninguna cámara. Esto se realiza con el propósito de poder saber en si la cantidad de memoria consumida por una sola cámara y así saber si el rendimiento era bueno, malo o regular.

Pues bien, al momento de agregar una sola cámara la memoria *RAM* ocupada por el *software* aumentaba entre 30 y 70 megas más, de forma variable, por lo que al final la memoria utilizada por el *software* y una sola cámara era de alrededor de 300 MB. Al aumentar la cantidad de cámaras, el software esta vez llego a soportar entre 20 y 25 cámaras antes de que alguna excepción o falla parará al programa.



Anexo 1.1 Prueba versión 2

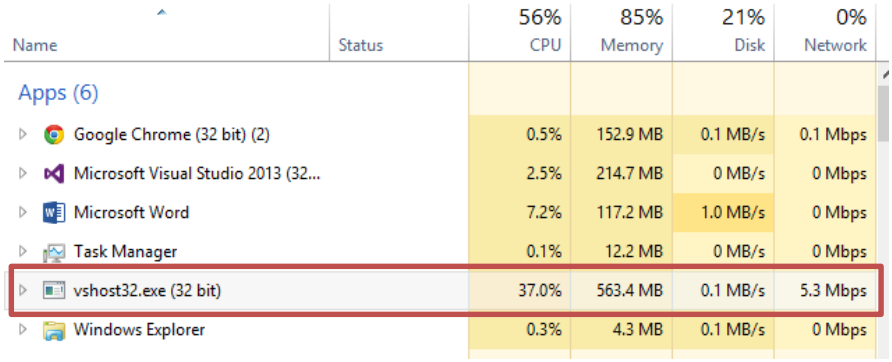
Es preciso también mencionar que, el rendimiento del software mejoro mucho con respecto al anterior, y que cuando se cargaban las 20 o 25 cámaras, el software podía pasar un par de días sin registrar error alguno, incluso hasta llego a durar funcionando de forma correcta durante una semana, después de esto una excepción volvía a aparecer.

En la tercera versión se realizaron las mismas pruebas que en la segunda versión, con el propósito de hacer una comparación, ya que esta versión fue realizada en *WPF*, mientras que la versión dos se realizó con *Windows Forms* y se deseaba comprobar si existían beneficios en el cambio de tecnología además de poder realizar animaciones más vistosas.

En este caso el consumo de recursos sin tener siquiera una sola cámara en funcionamiento es de tan solo 35 MB por lo que comparado con los 230 que consumía la versión anterior, esta es una muy buena mejora.

Ahora bien cuando el programa estaba utilizando una sola cámara el consumo de memoria crecía hasta alcanzar entre 70 y 110MB por lo que el consumo que tenía cada cámara era parecido al de la versión anterior.

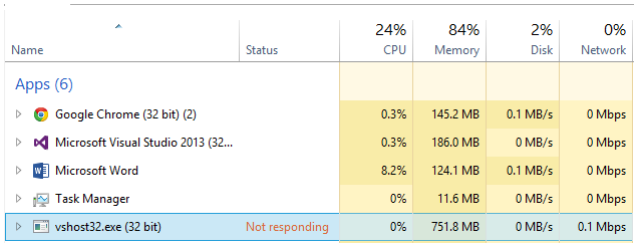
Lo destacable en este punto fue cuando se empezaron a agregar más cámaras a esta versión de cliente, para saber hasta qué momento dejaría de funcionar. Cuando alcanzo las 20 cámaras que fue más o menos el límite de la versión anterior, en esta versión tan solo se llegó a alrededor de 340MB, cuando la cantidad de cámaras llegó a 40 cámaras activas el consumo de memoria alcanzó los 600 MB (imagen x.x) y a estas alturas el software seguía funcionando de manera fluida, en este caso ya se tenían que tener dos pestañas abiertas.

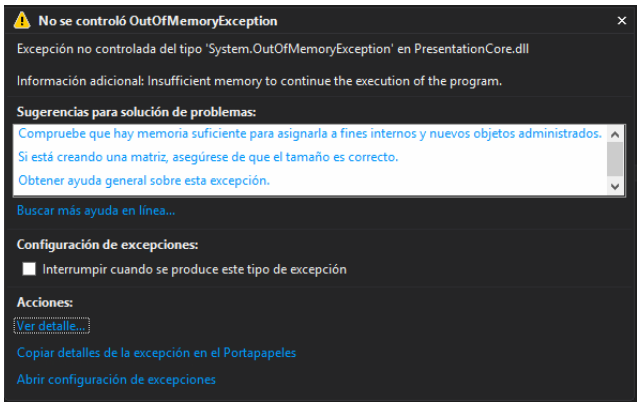


Name	Status	56% CPU	85% Memory	21% Disk	0% Network
Apps (6)					
Google Chrome (32 bit) (2)		0.5%	152.9 MB	0.1 MB/s	0.1 Mbps
Microsoft Visual Studio 2013 (32...		2.5%	214.7 MB	0 MB/s	0 Mbps
Microsoft Word		7.2%	117.2 MB	1.0 MB/s	0 Mbps
Task Manager		0.1%	12.2 MB	0 MB/s	0 Mbps
vshost32.exe (32 bit)		37.0%	563.4 MB	0.1 MB/s	5.3 Mbps
Windows Explorer		0.3%	4.3 MB	0.1 MB/s	0 Mbps

Anexo 1.2 Consumo de recursos con 20 cámaras

El programa dejo de funcionar cuando se agregaron 62 cámaras y se habían creado ya 4 pestañas, en ese momento la memoria supero los 750MB y se generó una excepción.





Anexo 1.3 Prueba Versión 3

Pero como ya se mencionó anteriormente el usuario que esté haciendo uso de este software no será capaz de visualizar 62 cámaras al mismo tiempo, mucho menos por el hecho de que cada pestaña del *software* solo es capaz de mostrar 20 cámaras, por lo tanto no valdría la pena que las demás cámaras estuvieran demandando recursos por lo que el consumo total del software en cuanto a memoria *RAM* se refiere sería de los 340MB mencionados, con lo que con las pruebas realizadas, se asegura al cliente que el *software* no fallara al menos por el consumo de los recursos y que aun así podrá visualizar tantas cámaras como desee simplemente seleccionando una pestaña diferente, con lo cual la pestaña que se encontraba activa y consumiendo recursos, pasara a un estado inactivo (sin consumo de recursos) y la pestaña actualmente seleccionada, pasara a ocupar los recursos que la pestaña anterior libero.

El realizar este tipo de pruebas me parece un aspecto muy importante en el desarrollo de *software* porque de esta manera es posible delimitar al *software* para que se desempeñe de forma correcta cuando esté siendo utilizado por el usuario final.