UNIVERSIDAD AUTONOMA DE MADRID

ESCUELA POLITECNICA SUPERIOR

SISTEMA INTELIGENTE DE GESTIÓN DE MÚLTIPLES CÁMARAS DE VIDEOSEGURIDAD

-PROYECTO FIN DE CARRERA-

Guillermo López-Oliva Santa Cruz

FEBRERO 2009

SISTEMA INTELIGENTE DE GESTIÓN DE MÚLTIPLES CÁMARAS DE VIDEOSEGURIDAD

AUTOR: Guillermo López-Oliva Santa Cruz
TUTOR: José María Martinez Sanchez

Video Processing and Understanding Lab (VPULab)
Dpto. de Ingeniería Informática
Escuela Politécnica Superior
Universidad Autónoma de Madrid
Febrero de 2009

Palabras clave:

Videoseguridad, *Streaming*, video procesado, cliente-servidor, detección de movimiento, adaptación de contenidos.

Resumen:

El objetivo de este Proyecto Fin de Carrera es el diseño y desarrollo de un sistema de videoseguridad inteligente distribuido, de acceso remoto para el usuario final (que es la persona que visualiza la grabación). Este sistema soportará múltiples cámaras de red, y admitirá cierta configuración por parte de un "supervisor" para ajustar la funcionalidad del sistema a sus propias necesidades. Mediante algoritmos de detección de movimiento el sistema será capaz de decidir de entre varias escenas cual es la relevante en cada instante y la mostrará por pantalla al usuario final, quien recibirá un *stream* de video comprimido servido por el sistema inteligente.

En resumen, la idea es crear una aplicación para vigilancia de múltiples zonas simultáneamente que facilite al operario de seguridad su labor.

Abstract:

The main objetive of this PFC is the design and development of an intelligent distributed surveillance system, providing remote access to the final user (which is the person who watches the recording). This system will control several network cameras, and will be configurable by a supervisor in order to adjust the system performance in a suitable form. By using some motion-detection algoritms the system will be capable of deciding which is the most relevant scene of the set of scenes and will display it to the final user, sending a compressed video stream to him.

To sum up, the main goal is to create an application for the surveillance of multiple areas at the same time in order to help the security guard doing his job.

Agradecimientos

Quiero agradecer a todas las personas que me han ayudado en este proyecto, de algún modo u otro:

A José María Martínez Sánchez, mi tutor, por ofrecerme la oportunidad de realizar este proyecto, y porque siempre tuvo un momento para ayudarme y atenderme cuando lo necesitaba.

A los miembros del VPULab, quienes estuvieron siempre dispuestos a echarme una mano, especiamente a Javier Molina, Juan Carlos San Miguel, Fernando López y Alvaro García Martin.

A todos los profesores de la carrera. Cada uno de ellos ha contribuído un poco en mi formación como ingeniero.

A mis amigos de la Universidad, con quienes además de pasar buenos ratos hemos superado juntos muchas asignaturas de la carrera.

Y a mis padres, por su interés y apoyo a lo largo de toda la carrera.

Sin ellos no habría sido posible la realización de este proyecto.

INDICE DE CONTENIDOS

INDICE DE FIGURAS

INDICE DE TABLAS

1 Introducción

Actualmente son pocos los edificios públicos y centros de la administración publica (incluidos estaciones de metro, aeropuertos, bancos, etc..) que no cuentan ya con sistemas que permitan garantizar la seguridad del interior e incluso de los alrededores de los mismos, que debido a su alta popularidad, no suponen un alto gasto a las empresas que deciden instalarlos.

Los sistemas habituales de vigilancia y seguridad solían caracterizarse por la existencia de múltiples cámaras de seguridad analógicas conectadas a un grupo de monitores supervisados por uno/varios vigilantes en una única sala, y en algunos casos tenían sistemas de grabación. Estos sistemas eran los llamados CCTV (Circuitos Cerrados de Televisión) y servían, por un lado para tener un registro visual de las incidencias y por otro lado para poder detectarlas en tiempo real (llevar a cabo detenciones, etc.). Dado que el operario debía vigilar varios escenarios simultáneamente y durante varias horas seguidas, la probabilidad de que pasaran por alto situaciones potencialmente peligrosas resultaba ser bastante alta.

Figura 1-1: Centro de videoseguridad

Gracias a la evolución tecnológica, este mundo ha mejorado sensiblemente de forma que las cámaras ahora son digitales, y disponen de una conexión IP, de forma que pueden tratarse como nodos o elementos de una red de comunicaciones. Esto permite que las cámaras transmitan video en modo *streaming* a un PC o servidor donde se almacena/visualizan las grabaciones, y además se procesa la información que recibe de éstas. De este modo, pueden crearse sistemas inteligentes de videoseguridad (*video surveillance*), que permitan facilitar la tarea de supervisión a los vigilantes de seguridad y que sean capaces de llamar su atención cuando se produce una situación peligrosa o extraña en el entorno que vigila y que permitan recuperar fácilmente la parte de la secuencia de vídeo relativa al evento que representa la razón por la que se le ha alertado.

El reciente interés en incluir sistemas de seguridad en locales comerciales, lugares públicos, escenarios militares, etc. crea esa necesidad de investigación y desarrollo de sistemas automáticos e inteligentes de videoseguridad. Actualmente, esta línea de investigación toca múltiples disciplinas como el tratamiento digital de señales, telecomunicaciones y estudios socio-culturales.

1.1 Motivación

La motivación de este PFC es crear un sistema de videoseguridad distribuido completo y efectivo, que comprenda un conjunto de múltiples cámaras de vigilancia, y que permita a su usuario final detectar rápidamente incidencias o eventos significativos simplemente visualizando únicamente una secuencia de video. Por ello, el sistema tendrá que producir esta secuencia en base a la imagen de la cámara relevante en cada instante de tiempo, en base a distintos algoritmos de decisión.

De este modo, un vigilante de seguridad podría utilizar este programa para supervisar varias áreas al mismo tiempo utilizando diferentes dispositivos: desde una sala mediante un PC de sobremesa, o desde un área con conexión inalámbrica, mediante un ordenador portátil, una PDA, un *PocketPC* o un *Smartphone*. Dicho vigilante podría configurar el sistema con diferentes algoritmos para que el cambio de cámara vaya inducido por diferentes eventos, por ejemplo: máximo movimiento (en la escena), ausencia de movimiento, presencia de objetos abandonados, etc. Igualmente estos eventos podrían combinarse entre ellos con distintos pesos o incluir una función de envejecimiento de la importancia del evento para evitar que una cámara acapare toda la atención en casos de similar actividad de los eventos seleccionados.

1.2 Objetivos

Para la creación de este sistema distribuido nos hemos propuesto dos objetivos (o dos etapas) principales. El primero es el desarrollo de la **aplicación inteligente multicámara**, que sea capaz de monitorizar simultáneamente la señal de varias cámaras de seguridad de manera simultánea, conectadas a la misma red local que el servidor (preferentemente una red Gigabit-Ethernet), procesar estas señales y clasificarlas por su importancia (siendo un criterio de importancia, por ejemplo, el movimiento de los objetos en la escena que captura cada cámara).

El segundo objetivo es la creación de un **sistema de *streaming* multiplataforma**. Esto es, un sistema que permite servir vídeo comprimido al usuario final, que podrá recibir y visualizar la aplicación cliente mediante algún programa reproductor de video comercial (como por ejemplo, WMP o VLC). El valor añadido de esta aplicación será la supervisión remota mediante no solo PC's de sobremesa, sino también dispositivos móviles (tanto ordenadores portátiles como PDA's y algunos teléfonos móviles).

El sistema final, que será el resultado de la combinación de las anteriores dos etapas, integrará diferentes tecnologías y plataformas, algunas desarrolladas por el VPULab de la Universidad Autónoma de Madrid como son DiVA (*Distributed Video*

Analysis)[7] y CAIN (*Content Adaptation INtegrator*)[8]. Para ello será necesaria una arquitectura cliente servidor que permita una eficiente cohesión entre los distintos módulos que lo componen.

1.3 Organización de la memoria

La memoria consta de los siguientes capítulos:

- **Capítulo 1**. Introducción, motivación y objetivos del proyecto
- **Capítulo 2**. Análisis del estado del arte en cuanto a videoseguridad, *streaming*, y dispositivos móviles.
- **Capítulo 3**. Descripción del diseño y arquitectura del sistema
- **Capítulo 4**. Descripción del desarrollo e implementación del sistema
- **Capítulo 5**. Integración, pruebas y resultados.
- **Capítulo 6**. Conclusiones obtenidas tras el desarrollo del sistema. Futuras líneas de desarrollo y mejoras del sistema posibles.
- **Anexo A**. Manual de instalación
- **Anexo B**. Manual de programación
- **Anexo C**. Estudio de los dispositivos PDA y sus Sistemas Operativos.

2 Estado del arte

2.1 Análisis de secuencias de videoseguridad

2.1.1 Introducción

Los sistemas inteligentes de videoseguridad [1] se basan en la monitorización de objetos (tanto móviles como estáticos, pues también podría interesar detectar objetos abandonados) que permanecen en una determinada escena. La principal misión de estos sistemas es proporcionar una interpretación automática de la/s escena/s en cuestión y realizar una predicción del comportamiento futuro de los objetos observados basada en la información adquirida por las cámaras. El procesado de la información en un sistema inteligente comprende las siguientes etapas: detección de objetos móviles, seguimiento de dichos objetos, y análisis de comportamiento en base a los datos recopilados. En el análisis de secuencias de videoseguridad, es necesario separar las secuencias en objetos móviles y estáticos (*foreground*) y fondo (*background*) [2].

Por tanto, dichos sistemas de *video surveillance* han de presentar una serie de características como son la facilidad de uso, la inclusión de algoritmos con un bajo coste computacional (han de poder funcionar en tiempo real), y que puedan ser adaptarse a las condiciones de la escena (mediante una adecuada configuración) que pueden cambiar durante el transcurso de largos periodos de tiempo

2.1.2 Revisión de técnicas de análisis de secuencias de video

El análisis de video en videoseguridad es un área de investigación permanente y actual. En particular, las áreas clave son la detección y seguimiento de objetos (*tracking),* identificación de personas y sistemas de *video surveillance* a gran escala [3]. La segunda área clave de la videoseguridad, la identificación de personas basada en video también ha sido un importante tema de estudio en los últimos años. El reconocimiento facial [4] ha sido una modalidad líder de identificación en sistemas de *video surveillance* industriales.

La actual línea de investigación en videoseguridad no solo apunta en la dirección de la detección de objetos y el seguimiento, sino también hacia la detección de eventos y la auto-calibración de sistemas. Desde la perspectiva de un análisis de la inteligencia humana, la interpretación de datos de análisis para la detección de eventos de interés es el reto más difícil en el desarrollo de aplicaciones de videoseguridad. El área de la interpretación de eventos basada en el contexto en una escena monitorizada [5] está actualmente bajo investigación. En esta área se incluyen: el modelado geométrico del entorno, de la actividad de los objetos, personas, vehículos, etc., desarrollo de técnicas de auto-adaptación para mejorar el rendimiento y detección de eventos inusuales y técnicas de adquisición información a múltiples escalas.

En la figura 2-1 se muestra un sistema donde las cámaras estáticas cubren por completo la escena de interés y aportan una visión general de la misma, mientras que las cámaras PTZ (*pan tilt zoom*) pueden obtener información detallada o a pequeña escala de objetos de interés en la escena.

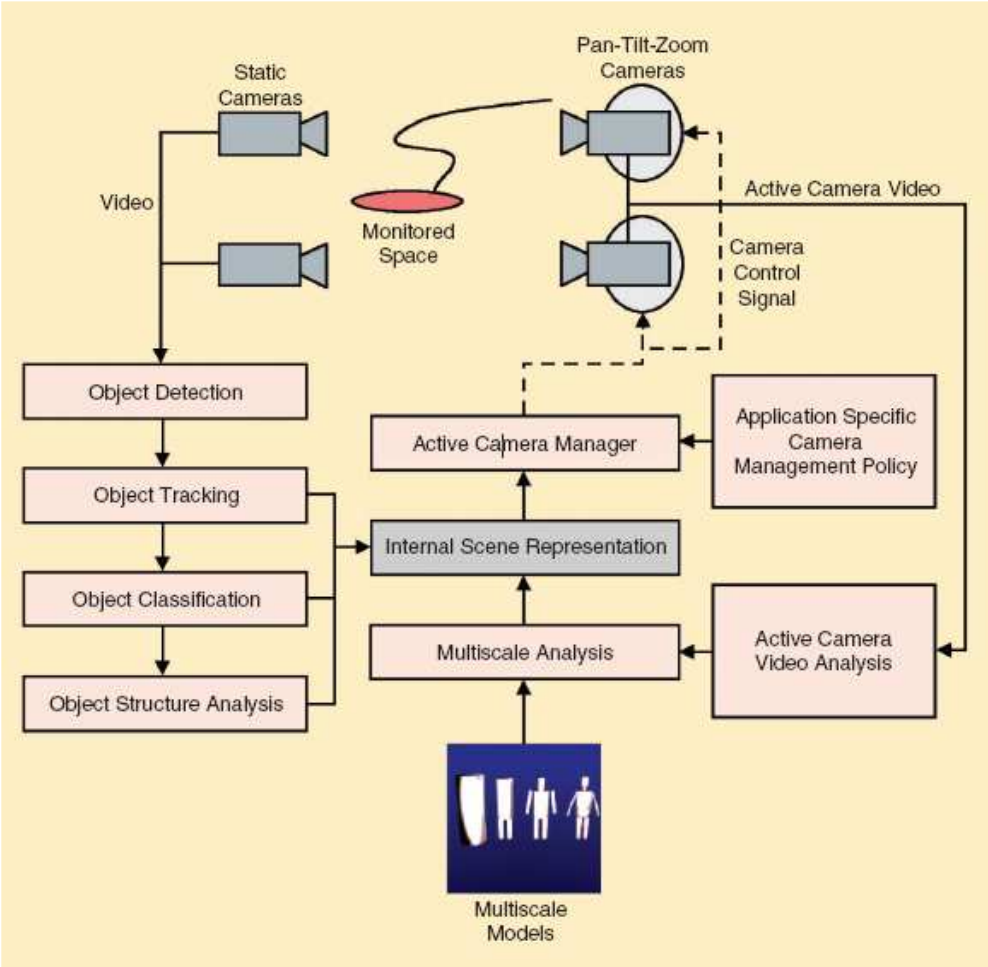


Figura 2-1: esquema de un sistema de video surveillance [5]

2.1.2.1 Detección de objetos

La detección de objetos es la primera etapa de la mayoría de sistemas de seguimiento de objetos y por ello requiere una especial mención. Existen dos principales enfoques de la detección de objetos [5]. El primero, llamado *background extraction* (Separación Fondo-Objeto), asume un fondo "estacionario" (sin cambios a lo largo del tiempo), y por tanto considera a todos los cambios en la escena como objetos de interés. El segundo, *salient motion detection* (Detección de Movimiento Principal) asume que en la escena pueden coexistir varios tipos de movimiento, de los cuales solo algunos son interesantes desde una perspectiva de videoseguridad.

La técnica *salient motion detection* tiene un buen rendimiento en entornos dinámicos, porque es muy adaptativa, sin embargo no es efectiva extrayendo todos los

píxeles de los objetos relevantes en la escena. Por otro lado, con la técnica *background extraction* se obtiene un mayor rendimiento en la extracción de información de los objetos aunque es muy vulnerable a cambios en el entorno. La técnica de Extracción de Fondo Adaptativa conlleva la creación de un modelo de *background* y la continua actualización del mismo para evitar errores en la detección cuando hay cambios en el entorno. Existen diferentes técnicas de modelado del fondo relacionadas con la aplicación a la que sirven. Por ejemplo, en entornos cerrados con una buena iluminación y cámaras fijas es posible crear un modelo de fondo sencillo comparando las imágenes de una secuencia adquiridas durante un breve periodo de tiempo. Las escenas de exteriores suelen tener una variabilidad alta en las condiciones del entorno, por eso es necesario tener modelos adaptativos del fondo aunque la robustez se traduzca en alto coste computacional.

2.1.3 Algoritmo de detección de movimiento

Una interesante aplicación en el área del procesado de video es el diseño y desarrollo de video-sensores. Un video-sensor no es mas que una herramienta de análisis de video que extrae información relevante de video sin comprimir (*raw video*). El uso de video-sensores aporta ventajas como bajo coste, buen rendimiento a lo largo del tiempo y protección de la privacidad. Estos sensores, por tanto, actúan como un filtro de situaciones o eventos interesantes en la escena que requerirían una evaluación minuciosa por parte de una persona. De este modo, a dicha persona se le presenta exclusivamente la información relevante (al menos, una gran parte del tiempo, dependiendo de lo efectivo que sea el sensor).

Anteriormente hemos visto que en el análisis de secuencias de video era necesaria la separación del *foreground* y *background* para la detección de objetos en movimiento. Sin embargo puede hacerse un análisis mucho más sencillo (como el detallado en [6]), con vistas a obtener información de la escena mediante detección de movimiento. Además, teniendo en cuenta que nuestra aplicación ha de clasificar múltiples señales de video que representan distintas escenas, de este modo tenemos un criterio “cuantitativo” que nos facilita la tarea.

La detección de movimiento (*motion detection*) puede aplicarse en la imagen entera o en una Región de Interés (*R*). Sea *N* el número de píxeles en *R* y sea *I(t , p)* el valor de la imagen en el instante *t* en la posición *p* (el píxel), entonces una simple medida de la distancia entre dos imágenes consecutivas en el tiempo puede ser:

$$D(t) = \frac{1}{N} \sum_{p \in R} \left| I(t, p) - I(t-1, p) \right|$$

En ausencia de movimiento:

$$I(t, p) = I(t-1, p)$$
$$D(t) = 0$$

No obstante, en imágenes de video siempre hay presente ruido, con lo cual, un modelo más preciso en ausencia de movimiento sería:

$$I(t, p) = I(t-1, p) + n(p)$$

donde *n(p)* es un ruido con media nula, con varianza σ_n^2. En este caso, *D(t)* sería un proceso estocástico con los siguientes estadísticos:

$$\overline{D} = k_1 \sigma_n$$

$$\sigma_D^2 = k_2 \frac{\sigma_n^2}{N}$$

donde k_1 y k_2 son constantes que dependen del tipo de función de densidad de probabilidad (*pdf*) n*(p)*. Si *n(p)* es una gaussiana:

$$k_1 = \frac{2}{\pi}$$

$$k_2 = \frac{\pi - 2}{\pi}$$

Podemos establecer un umbral (*threshold*) en *D(t)* para la detección de movimiento. El valor del umbral debe establecerse acorde al nivel de ruido σ_n^2:

$$T = \overline{D} + k_3 \sqrt{\sigma_D^2} = \sigma_n \left(k_1 + k_3 \sqrt{\frac{k_2}{N}} \right)$$

donde k_3 depende de la probabilidad de falsa alarma deseada. Nótese que independientemente de el tipo de *pdf* de *n(t)*, en ausencia de movimiento, para un *N* suficientemente grande, *D(t)* es aproximadamente gaussiana, de acuerdo con el Teorema del Límite Central.

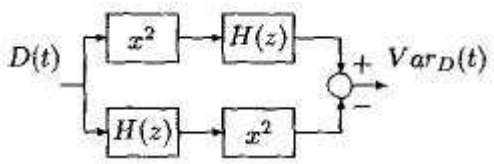
En la practica, el valor de σ_n^2 es desconocido y debe estimarse cuando hay ausencia de movimiento entre imágenes consecutivas en el tiempo. Esto puede llevarse a cabo en una etapa de preparación previa (*setup*) donde:

- Nos aseguraremos de que no haya movimiento al activar el sensor.
- Ajustaremos los niveles de iluminación en la escena, dado que el nivel de ruido depende de éstos (aumenta cuando la iluminación es baja).

Una posible estrategia es actualizar el *threshold* cuando no hay movimiento en la escena (esto requiere, por tanto, saber cuándo ocurre esto). Sin embargo, como hemos remarcado antes, en ausencia de movimiento y con un número de píxeles grande *D(t)* tiene una varianza muy pequeña y puede ser un factor que nos determine cuándo actualizar el umbral.

La varianza puede estimarse mediante un filtro paso bajo *H(z)* de ganancia unidad para la componente continua:

$$H(z) = \frac{1-a}{1-az^{-1}}$$



Debemos, por tanto ajustar el parámetro *a* que controla la duración de la respuesta al impulso y determina el número de muestras utilizado para el cálculo de la varianza y el umbral de la varianza, T_V.

Desearemos que se cumplan dos propiedades para el filtro: respuesta rápida y alta sensibilidad. Para una rápida respuesta consideraremos *D(t)* como una secuencia del tipo:

$$D(t) = \begin{cases} C, t < t_0 \\ C+1, t = t_0 \end{cases}$$

Queremos el valor de a que maximiza la varianza de *D(t):*

$$Var_D(t_0) = a - a^2$$

por tanto,

$$\max(\ Var_D(t_0)) = 0.5$$

Para la segunda propiedad, entendemos que cuando hay movimiento en *D(t)* hay variaciones mucho mayores que cuando no lo hay, por tanto consideramos *D(t)* del siguiente tipo:

$$D(t) = \begin{cases} C, t\_par \\ C+1, t\_impar \end{cases}$$

La varianza ahora es:

$$Var_D(t) = 0.25\left(1-\left(\frac{1-a}{1+a}\right)^2\right),$$
$$\max(Var_D(t)) = 1$$

Por lo tanto, el valor deseado ha de estar entre 0.5 y 1. El punto de cruce de las dos varianzas (representadas como funciones del parámetro *a*) se produce en:

$$a_{opt} = \frac{\sqrt{5}-1}{2} = 0.618$$

Ahora ajustaremos el umbral de la varianza que nos permitirá determinar la existencia de movimiento. Así como el valor de *D(t)* puede ser más o menos grande dependiendo del ruido, dado que el número de píxeles *N* será grande en la región *R* , *D(t)* presentará pequeñas variaciones. Por ello, es comprensible que las variaciones de esta función cuando se produzca movimiento sean notables.

Si establecemos un límite superior a σ_n^2 ($\sigma_{n,\max}^2$) la media de $Var_D(t)$ en ausencia de movimiento será:

$$\overline{Var_D(t)} < \sigma_{n,\max}^2 \frac{k_2}{N}$$

en otras palabras, la varianza de $Var_D(t)$ depende del momento de orden cuarto de la señal de ruido *n(t)*.

Para evitar tener que calcular estos momentos estadísticos se ha obtenido un valor experimental que permite obtener altas tasas de detección, prácticamente sin “falsas alarmas”, que evita tener que ajustar parámetros para el correcto funcionamiento del video-sensor. Podemos usar, por tanto, como valor práctico de *threshold*:

$$T_V = \frac{500}{N}$$

Para resumir, hemos visto que cuando no hay movimiento entre dos cuadros consecutivos, la medida de “similitud” entre ellos tiende a ser constante a lo largo del tiempo, mientras que cuando este existe, esta suele presentar grandes variaciones.

Hemos comentado uno de los muchos algoritmos que existen para la detección de movimiento. Éste en particular presentaba la cualidad de la sencillez y bajo coste computacional. Existen muchos otros más precisos que realizan segmentación (para más información consultar la web

http://iris.usc.edu/Vision-Notes/bibliography/contentssegment.html#2-D%20Region%20Segmentation%20Techniques,%20Snakes).

2.2 Sistemas distribuidos de videoseguridad

2.2.1 Evolución de los sistemas de videoseguridad

La evolución histórica de los sistemas de video vigilancia [1] comienza con los CCTVs analógicos, que comprendían un set de cámaras conectadas cada una a un set de monitores localizados en una única sala de control supervisada por operarios (conformando una "Matriz de video", ver figura 2-3). Las cámaras convencionales de seguridad en los sistemas CCTV utilizan generalmente CCDs (*charge coupled device*) para la captura de imágenes. La imagen digital obtenida de los sensores se convierte en una señal de video compuesto analógica y es transmitida a la matriz de monitores a través, generalmente, de cables coaxiales. La conversión analógico-digital acarrea una degradación de la imagen, y por otro lado la señal analógica es vulnerable al ruido que puede introducirse durante la transmisión. Sin embargo, es posible encontrar sistemas CCTV que aprovechan la imagen digital capturada procesada por ordenadores de altas prestaciones.

La mejora tecnológica en estos sistemas dio lugar al desarrollo de sistemas semi-automáticos, también conocidos como sistemas de video--vigilancia de segunda generación, que incluyen algoritmos de detección automática en tiempo real que ayudan al usuario a detectar eventos relevantes.

Figura 2-2: Matriz de monitores

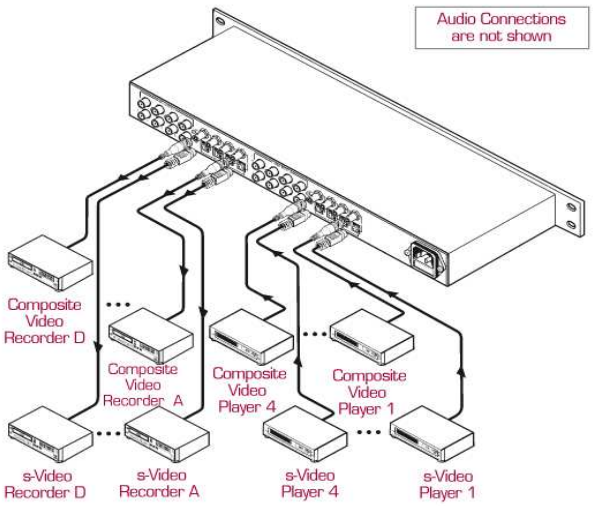


Figura 2-3: Matriz de video CCTV (http://www.kramerelectronics.com)

A continuación se resume la evolución tecnológica de los sistemas inteligentes [1]:

- **1ª Generación:**

→ Sistemas CCTV analógicos, con un alto rendimiento en la mayoría de las situaciones.
→ Problemas: el uso de sistemas analógicos para distribución y grabación.
→ Investigación actual: grabación de video digital y compresión de video.

- **2ª Generación:**

→ Sistemas que combinan CCTVs y técnicas de videoseguridad automática, que refuerzan la eficiencia en las tareas de vigilancia.
→ Problemas: necesidad de algoritmos robustos de detección y seguimiento.
→ Investigación actual: algoritmos de visión artificial en tiempo real, análisis de variabilidad de la escena y de patrones de comportamiento.

- **3ª Generación:**

→ Sistemas automáticos de videoseguridad distribuidos, que pueden combinar diferentes tipos de sensores para obtener información más precisa.
→ Problemas: problemas de integración y de comunicación, debidos a la distribución; la metodología del diseño de sistemas y la movilidad de las plataformas.
→ Investigación actual: inteligencia artificial en sistemas distribuidos, desarrollo de plataformas con razonamientos probabilísticos, técnicas de vigilancia de múltiples cámaras, calibración automática de sistemas.

Figura 2-4: sistema CCTV actual monitorizado en un ordenador personal.

2.3 Sistemas de streaming de video comprimido

2.3.1 Introducción

El término s*treaming* (más específicamente, *online streaming*) se refiere al medio de distribución de contenidos multimedia a través de la red bajo la demanda de los usuarios (clientes). Ésta tecnología permite además la visualización de los contenidos en tiempo real, es decir, mientras el usuario los está descargando. De otro modo, sería necesario tener el contenido completo del archivo descargado para más tarde poder reproducirlo.

Generalmente el modo de funcionamiento de un sistema de *streaming* es el siguiente. En primer lugar el cliente conecta con el servidor de *streaming*, y si la identificación es satisfactoria comienza el envío de los contenidos. El cliente comienza a recibir los datos y los introduce en un buffer. El cliente puede comenzar la reproducción del contenido mientras el buffer se va llenando. En caso de descensos de la velocidad de transmisión o incluso caídas de la red, se sigue reproduciendo el contenido un tiempo determinado hasta que el buffer finalmente se vacía, momento en el que la reproducción se cortaría también.

Desde el punto de vista del cliente de contenidos, existen dos principales "modelos de entrega" [7]: "***download*"** (descarga) donde el cliente comienza a reproducir el contenido después de haberlo recibido por completo desde el servidor; y "***streaming",*** donde el contenido se va reproduciendo mientras se reciben los datos. Habitualmente los servidores de streaming tienen dos métodos de entrega:

- **En directo** (*Live*)**:** difusión de eventos en tiempo real. Este tipo de streaming es útil cuando el cliente espera recibir video tan pronto como éste esté disponible. Mediante software de difusión en red se puede dar soporte a eventos en tiempo real, video conferencias y sistemas de videoseguridad a través de Internet. Este software codifica el video desde la fuente (cámara, televisión digital, etc.) y la envía al servidor de streaming. Después el servidor "sirve" o "refleja" este video en directo a los clientes.
- **Bajo demanda** (*On demand*): para entrega de contenidos multimedia bajo demanda (VoD), como películas o repeticiones deportivas, en donde cada usuario puede iniciar la reproducción desde el principio o desde donde quiera (pues se transmite un stream a cada cliente por separado).

Actualmente, el modo de acceso más popular a contenidos multimedia vía *streaming* es mediante programas que se instalan en los navegadores de Internet como *plugins* (como *applets*) capaces de reproducir los contenidos multimedia codificados con un formato determinado. Por ejemplo, la mayoría de páginas web oficiales de emisoras de radio ofrecen la posibilidad de escuchar en directo sus emisiones, por medio de un *plugin* reproductor. El ejemplo más extendido actualmente de *streaming* de video es *Youtube*¸ que permite a los usuarios visualizar videos almacenados en sus servidores bajo demanda, y requiere también la instalación de un *plugin* reproductor en el navegador web.

Los protocolos RTSP (*Real-Time Streaming Protocol*), RTP (*Real-time Transport Protocol*) y RTCP (*Real-time Transport Control Protocol*) fueron diseñados específicamente para el envío de contenidos multimedia a través de la red (los dos primeros funcionan sobre UDP). Los protocolos fiables como TCP (*Transmission Control*

Protocol) garantizan la entrega de todos los paquetes del flujo multimedia, sin embargo, asumiendo pérdidas de los datos en la red, las continuas retransmisiones impiden la reproducción en tiempo real de los contenidos.

A continuación presentaremos una serie de plataformas comerciales conocidas que permiten el *streaming* de contenidos multimedia, y particularmente, de video comprimido.

2.3.2 Apple (http://www.apple.com/server/macosx/)

Esta conocida marca de ordenadores y creadora del sistema operativo MAC OS, ofrece varias soluciones para instalar un sistema de *streaming* multimedia.

2.3.2.1 QuickTime Streaming Server:

Mediante el estándar abierto RTP/RTSP, QuickTime Streaming Server permite al usuario ofrecer contenido pregrabado o en directo por Internet. Con *Instant-On*, la innovadora tecnología de emisión pendiente de patente de Apple, el contenido empieza a reproducirse tan pronto como se pulsa el enlace: no es necesario esperar a que el archivo se descargue.

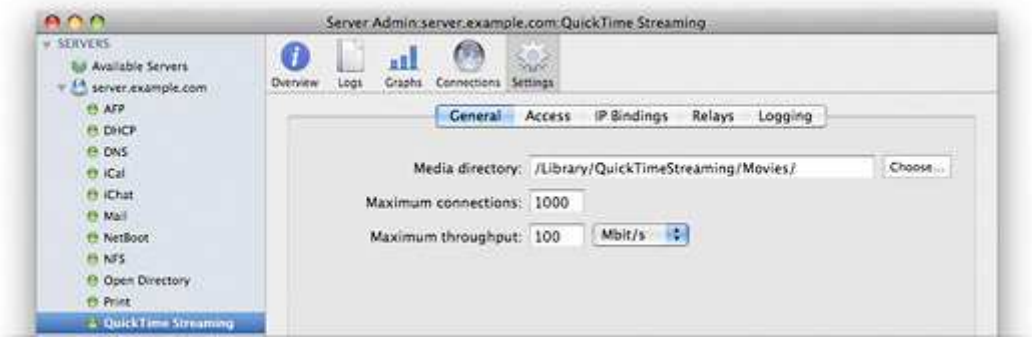


Figura 2-5: QuickTime Streaming Server

La figura 2-6 muestra una configuración con QTSS para transmisión en directo de video. El ordenador "*broadcaster*" con Quick Time Broadcaster captura y codifica el video y el audio provenientes de la cámara. La señal codificada es enviada por una red IP al equipo servidor que ejecuta Quick Time Streaming Server. Esta aplicación envía esta señal a través de Internet o a través de la red local del servidor a equipos "clientes" que demanden el contenido por medio de Quick Time Placer.

QuickTime es totalmente compatible con los estándares multimedia mundiales más recientes, entre ellos H.264, MPEG-4 y 3GPP. Igualmente, QuickTime Streaming Server es compatible con la emisión de archivos H.264, MPEG-4 y 3GPP de forma nativa, para que el contenido pueda disfrutarse con cualquier reproductor multimedia estándar para Mac o Windows, así como con numerosos dispositivos compatibles como móviles y descodificadores digitales. También permite ofrecer archivos a clientes MP3 estándar mediante protocolos compatibles con Icecast sobre HTTP.

Mac OS X Server Leopard incluye QuickTime Streaming Server 6, la última versión del potente servidor de streaming Apple. Esta actualización da un mayor apoyo para el streaming estándar de la industria de archivos 3GPP incluyendo la versión 6 para asegurar un buen flujo de streaming en momentos de congestión de la red. QTSS 6 también proporciona integración Open Directory que permite restringir el acceso a las secuencias utilizando los usuarios y grupos en el Mac OS X Server directorio LDAP. Leopard Server incluye una versión de 64 bits de QuickTime Streaming Server.

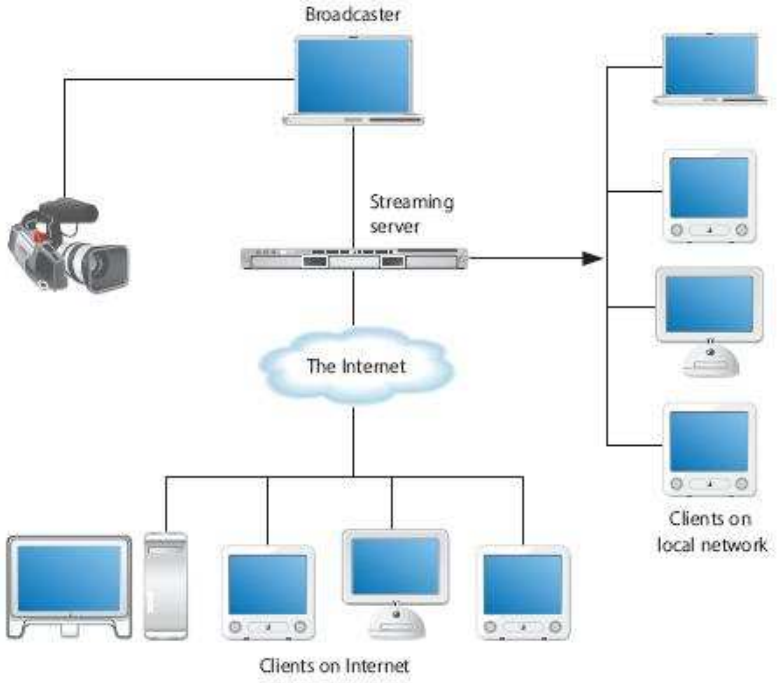


Figura 2-6: Esquema de funcionamiento del servidor de streaming

Requisitos del servidor

→ Mac OS X Server v10.5 (o posterior)

→Servidor Mac o equipo de sobremesa con un procesador Intel, PowerPC G5, o PowerPC G4 (867MHz o más rápido) procesador, 1 GB de RAM física; 20GB de espacio disponible en disco.

2.3.2.2 Darwin Streaming Server
(http://developer.apple.com/opensource/server/streaming/index.html)

Este servidor es la versión "*open source*" (código abierto) del servidor QTSS de Apple, que permite también la distribución de contenidos multimedia a través de Internet mediante los protocolos RTP y RTSP. Basándose en el mismo código base que QTSS, Darwin Streaming Server proporciona un alto nivel de configuración y funciona en una amplia variedad de plataformas permitiendo al programador modificar el código según sus necesidades.

Darwin Streaming Server es un proyecto de código abierto pensado para desarrolladores que hacer *streaming* de contenido QuickTime y MPEG-4 en plataformas alternativas a MAC OS X server, como son Windows, Linux y Solaris o para desarrolladores que necesitan ajustar el funcionamiento del servidor a sus necesidades. Apple no ofrece soporte técnico al Darwin Streaming Server, es la comunidad *open source community* la encargada de ello.

Aunque por defecto DSS es (al igual que QTSS) un servidor de video, puede configurarse para transmitir mp3 mediante el MP3Broadcaster que forma parte de DSS. También es posible la descarga de los archivos mediante el módulo QTSSHttpFileModule.

Tanto DSS como QTSS se basan su núcleo en un servidor que proporciona alta QoS (calidad de servicio) con características como Skip protection (protección ante interrupciones) e Instant-On (arranque instantáneo), y soporte a los formatos multimedia más recientes, MPEG-4 y 3GPP.

2.3.3 VideoLAN (http://www.videolan.org/)

VideoLAN es un proyecto de software abierto que produce software abierto gratuito de video, publicado bajo la licencia pública GNU. La solución de *streaming* de VideoLAN incluye dos programas:

- VLC media player que puede ser usado tanto como servidor como cliente para enviar y recibir contenidos multimedia por la red. VLC es capaz de transmitir todo lo que pueda leer (incluso entradas de video por tarjetas capturadotas, webcams, etc).
- VLS (VideoLAN Server), es capaz de transmitir archivos MPEG-1, MPEG-2 y MPEG-4, DVDs, canales de TVD satélite y terrestre, y videos en directo por la red tanto en unicast como multicast. La mayoría de la funcionalidad de VLS actualmente puede encontrarse en VLC (de hecho se recomienda el uso de éste último, pues VLS es un proyecto sin mantenimiento desde 2005). Además VLS no tiene capacidades de transcodificación.

La siguiente figura 2-7 muestra la arquitectura de *streaming* VideoLAN:

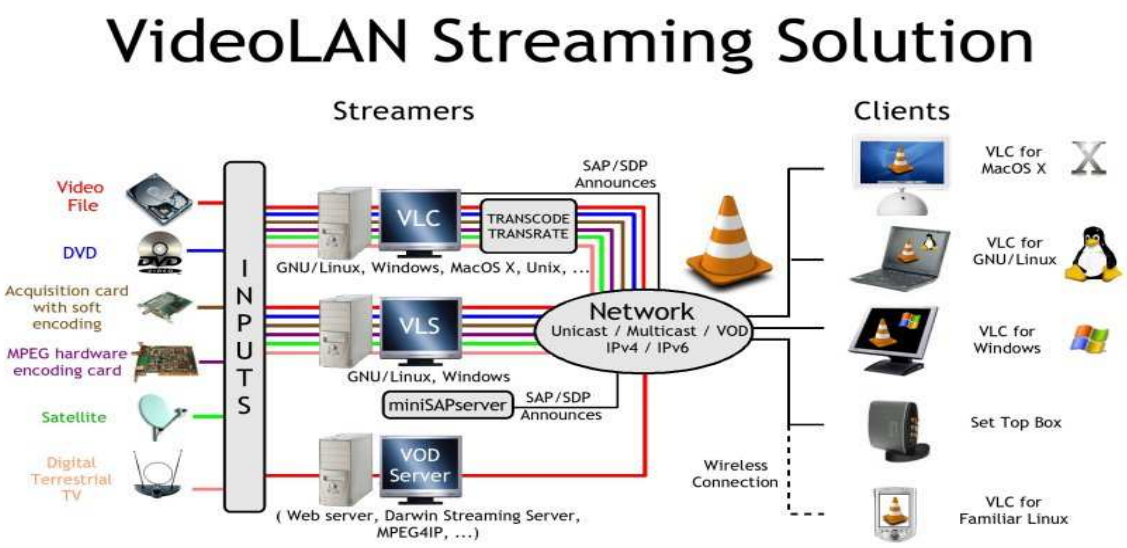


Figura 2-7: arquitectura de streaming VideoLAN

VLC media player funciona en una gran cantidad de plataformas: Linux, Windows, Mac OS X, BeOS, *BSD, Solaris, Familiar Linux, Yopy/Linupy y QNX. Puede leer contenidos con múltiples formatos y contenedores, dependiendo de la plataforma. En la URL de VideoLAN podemos encontrar todas las características de la plataforma, como los formatos de streaming, los codecs soportados, los filtros de procesado de imagen utilizados, etc. La siguiente tabla muestra los formatos multimedia de entrada admitidos:

Input media	UDP/RTP Unicast	✓	✓	✓	✓	✓	✓
	UDP/RTP Multicast	✓	✓	✓	✗	✓	✓
	HTTP / FTP	✓	✓	✓	✓	✓	✓
	MMS	✓	✓	✓	✓	✓	-
	TCP/RTP Unicast	✓	✓	✓	✓	✓	✓
	DCCP/RTP Unicast	✗	✗	✓	✗	✗	?
	File	✓	✓	✓	✓	✓	✓
	DVD [1]	✓	✓	✓	✓	✓	-
	VCD	✓	✓	✓	✗	✓	-
	SVCD [2]	◒	◒	◒	✗	◒	-
	Audio CD (without DTS)	✓	✓	✓	✗	✓	-
	DVB (Satellite, Digital TV, Cable TV)	✓	✗	✓	✗	✗	-
	MPEG encoder [3]	✓	✗	✓	✗	✗	-
	Video acquisition	✓ Direct Show	✗	✓ V4L, V4L2	✗	✗	? V4L, V4L2
Input formats	MPEG (ES,PS,TS,PVA,MP3)	✓	✓	✓	✓	✓	✓
	ID3 tags	✓	✓	✓	✓	✓	✓
	AVI	✓	✓	✓	✓	✓	✓
	ASF / WMV / WMA	✓	✓	✓	✓	✓	✓
	MP4 / MOV / 3GP	✓	✓	✓	✓	✓	✓
	OGG / OGM / Annodex	✓	✓	✓	✓	✓	✓
	Matroska (MKV)	✓	✓	✓	✓	✓	✗
	Real	✗	✗	✗	✗	✗	✗
	WAV (incuding DTS)	✓	✓	✓	✓	✓	✓
	Raw Audio: DTS, AAC, AC3/A52	✓	✓	✓	✓	✓	✓
	Raw DV	✓	✓	✓	✓	✓	✓
	FLAC	✓	✓	✓	✓	✓	✓
	FLV (Flash)	✓	✓	✓	?	✓	?
	Standard MIDI	✓	✓	✓	✓	✓	✓
	Creative™ Voice	✓	✓	✓	✓	✓	✓

Figura 2-8: capacidades de streaming de VideoLAN

2.3.4 Windows Media (www.microsoft.com/windows/windowsmedia/9series/server.aspx)

Microsoft también nos plantea una solución robusta para crear un sistema de streaming. Aunque podemos montar una plataforma de streaming mediante un *Web Server,* Microsoft nos aconseja utilizar Windows Media server. Esta es habitualmente la mejor opción de para transmitir contenidos multimedia debido a que Windows Media Services está específicamente diseñada para ello, y todos los componentes de Windows Media (WM Encoder, WM Services y WM Player) funcionan juntos para aumentar la calidad del servicio.

Las características principales del servidor de Windows Media son las siguientes:

- **Streaming a través de la mayoría de *firewalls***: mediante la opción de streaming con protocolo HTTP.
- **Streaming de contenido con derechos de autor (DRM):** Windows Media Rights Manager ayuda a los proveedores de contenidos a distribuirlos por Internet.
- **Rápidos envíos**: WM Services permite combinar streaming con descarga y descarga en caché para aumentar la QoS percibida por el usuario. Algunos servicios son: Fast Start, que descarga los primeros segundos del contenido lo más rapido posible de forma que se pueda empezar a reproducir lo antes posible; Fast Cache, que emplea el ancho de banda disponible para almacenar el máximo contenido posible; Fast Recovery, que envía datos de corrección de errores con los paquetes en vez de esperar a que ocurran dichos errores; y Fast Reconnect, que permite que el servidor restaure automáticamente las conexiones perdidas.
- **Streaming sin descarga (mediante Fast Cache)**
- **Broadcast (live)**. Los servidores Web solo pueden ofrecer contenido bajo demanda (archivos). El servidor de Windows Media puede ofrecer contenido en tiempo real mediante *Broadcasting* (como por ejemplo radio y TV).
- **Streaming Inteligente:** mediante la interacción de WM Server y WM Player para optimizar el envío bajo determinadas condiciones de ancho de banda.
- **Optimizado para transmisión de contenido Windows Media.** Esto son WMVideo y WMVAudio, y también JPEG y MP3.
- **Indexado**. Esto permite al usuario final rebobinar y avanzar a través del archivo mientras se envía (requiere interacción mediante WMServer y WM Player).
- **Administración de usuarios**. El servidor permite controlar el acceso de los usuarios al contenido, y monitorizar la actividad global del sistema en tiempo real.

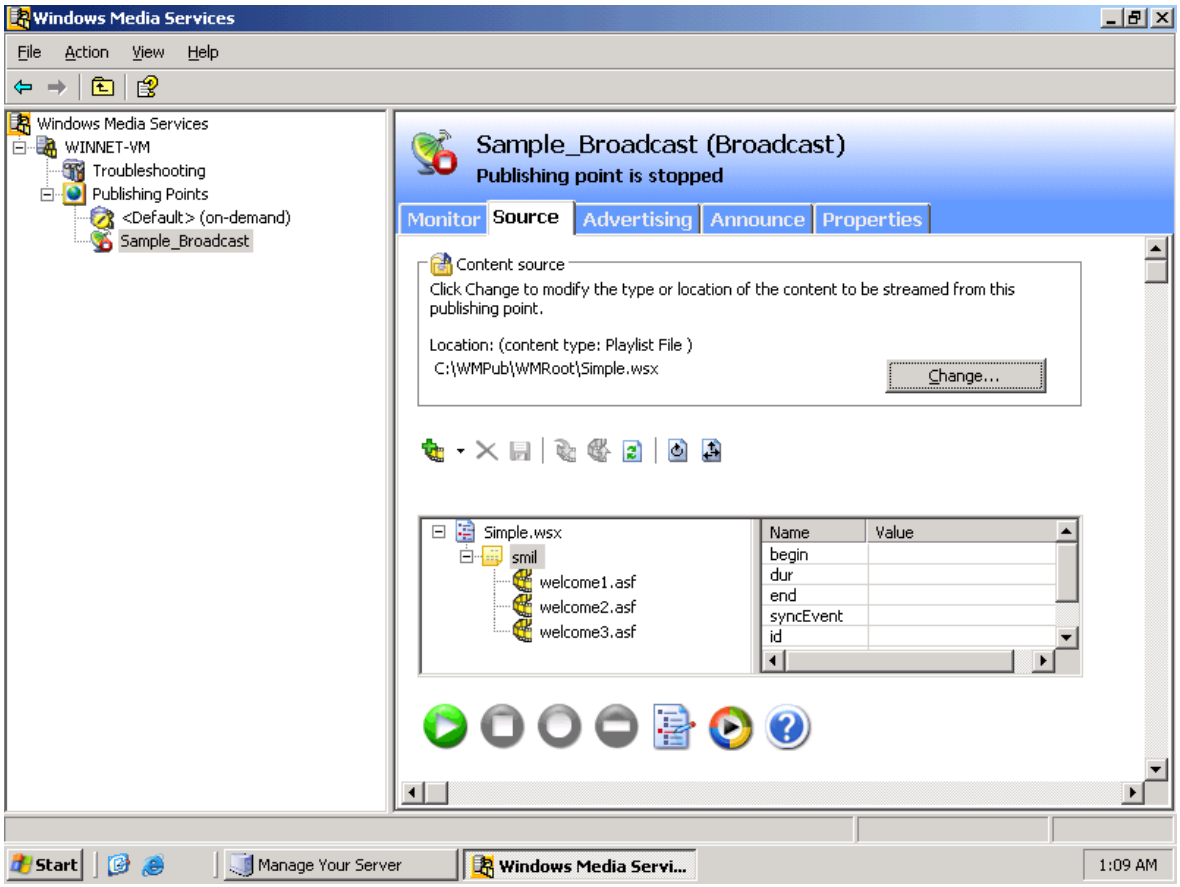


Figura 2-9: pantalla de configuración de Windows Media Services

WM Services soporta tres protocolos diferentes streaming en unicast: *Real Time Streaming Protocol* (RTSP), *Hypertext Transfer Protocol* (HTTP), y *Microsoft Media Server protocol* (MMS). Tanto RTSP y MMS pueden operar sobre TCP o UDP. HTTP únicamente funciona sobre capa de transporte TCP. WM Services también ofrece servicios de multicast.

En la siguiente figura se nos muestra un esquema de la arquitectura de Windows Media Services.

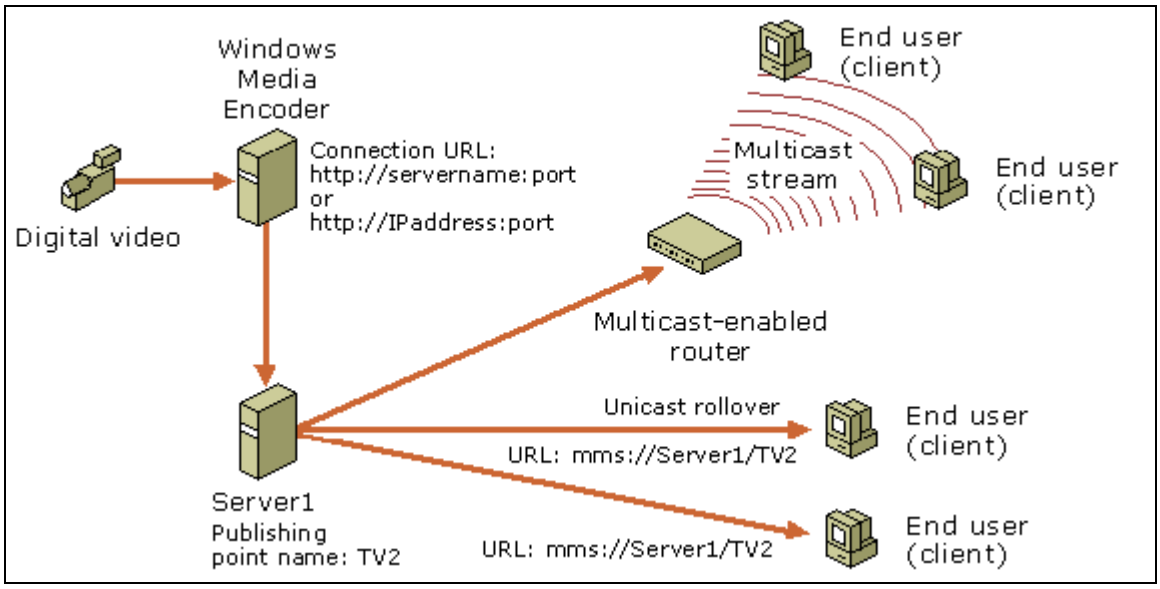


Figura 2-10: Streaming con Windows Media

La última versión a fecha de hoy es Windows Media Services 9 series, para Windows Server 2003 con Service Pack 1, y las nuevas versiones x64 de Windows Server 2003.

2.4 Dispositivos PDA

2.4.1 Introducción

Actualmente podemos encontrar en el mercado múltiples tipos de dispositivos móviles con capacidades de procesamiento y almacenamiento similares a las que hace no muchos años tenían los ordenadores personales de la época. Hablamos de los *Pocket PC*, las PDAs, los *smartphone* (teléfonos móviles de altas prestaciones), etc. Debido a ello, y a la creciente penetración de estos aparatos en el mercado, es razonable pensar que la migración de ciertas aplicaciones y servicios de uso habitual en PCs tanto domésticos como de empresa a este tipo de terminales será (y es) necesario en un futuro no muy lejano.

Un *Pocket PC* (ordenador de bolsillo), también llamado PDA (*Personal Digital Assistant*), es la especificación hardware para ordenadores de bolsillo que utilizan el sistema operativo *Microsoft Windows Mobile* (inicialmente *Windows CE*). Tienen también la capacidad de funcionar alternativamente con *Linux*, *NetBSD* y recientemente *Android* (de *Google*), aunque el término *Pocket PC* se aplica solo a dispositivos que funcionan con sistema operativo de Microsoft (que será en los que nos centraremos en este PFC). Se caracterizan por presentar una gran parte de las capacidades de los actuales PCs.

La mayoría de estos cuentan con una pantalla táctil como interfaz primaria con el usuario, puertos de entrada USB y ranuras para tarjetas de memoria, conectividad mediante IrDA (infrarrojos), Bluetooth y/o WiFi, etc. Además, muchos de estos dispositivos incluyen funcionalidades de telefonía móvil (*smartphones*), receptores de GPS, cámaras digitales, lectores de códigos de barras, etc.

El sistema operativo de las PDAs suele incluir calendarios, editor de notas, listas de contactos, soporte para correo electrónico y navegador web. Los *Pocket PC*, además llevan incluido el paquete Office que permite la edición de textos en *Word*, tablas de *Excel* y presentaciones de *PowerPoint*. Existen además en este momento miles de aplicaciones para dispositivos *Pocket PC* de las cuales muchas de ellas son “software abierto”.

En las siguientes subsecciones veremos aspectos de las PDA como su historia y evolución, los usos de las mismas y sistemas operativos que soportan. En el Anexo C podemos obtener más información acerca de la programación de aplicaciones multimedia para PDAs.

2.4.2 Historia

En 1989, el Atari Portfolio, fue una muestra temprana de algunos de los más modernos dispositivos electrónicos. Le siguieron otros dispositivos como los Psion Organiser el Sharp Wizard o la Amstrad Penpad que fueron sentando la base de las funcionalidades de las PDAs. No obstante la primera mención formal del término y concepto de PDA (*Personal Digital Assistant*) es 1992 en la presentación del *Apple* Newton, en una feria de electrónica en Las Vegas (EE.UU.). Sin embargo fue un sonoro fracaso financiero para la compañía *Apple*, dejando de venderse en 1998. La tecnología estaba aún poco desarrollada y el reconocimiento de escritura en la versión original era bastante impreciso, entre otros problemas. Aun así, este aparato ya contaba con todas las

características de la PDA moderna: pantalla táctil, conexión a una computadora para sincronización, interfaz de usuario especialmente diseñada para el tipo de máquina, conectividad a redes vía módem y reconocimiento de escritura.

En 1995 con la aparición de la empresa *Palm* comenzó una nueva etapa de crecimiento y desarrollo tecnológico para el mercado de estos dispositivos. Tal fue el éxito que las PDA son a veces llamadas *Palm* o *Palm Pilot*, lo cual constituye un caso de una marca registrada que se transforma en el nombre genérico del producto.

Figura 2-11: Apple Newton

Figura 2-12: Palm Pilot

Figura 2-13: Smartphone de HTC actual

La irrupción de *Microsoft Windows CE* (2000) y *Windows Mobile*(2003) en el sector los dotó de mayores capacidades multimedia y conectividad, y sobre todo incorporó a un público ya acostumbrado al uso de sus programas y que se los encontraban en versión reducida.

La irrupción de los s*martphones* (híbridos entre PDA y teléfono móvil) trajeron por un lado nuevos competidores al mercado y por otro incorporaron al usuario avanzado de móviles al mercado. De paso supuso la vuelta de un sistema operativo que había abandonado el mercado de las PDAs y ordenadores de mano en favor de los móviles : el *Symbian OS*. Las PDAs de hoy en día traen multitud de comunicaciones inalámbricas (Bluetooth, Wi-Fi, IrDA, GPS...) que los hace tremendamente atractivos hasta para cosas tan inverosímiles como su uso para domótica o como navegadores GPS.

En la actualidad se venden aproximadamente 4 millones de PDAs básicas (sin teléfono) al año, mientras que las ventas de *smartphones* alcanzan los 60 millones de unidades al año (datos de 2007). De acuerdo con un estudio de mercado de la empresa Gartner, el mercado de las PDAs en el tercer cuatrimestre de 2005 se presentaba de la siguiente forma:

- *Palm OS* de *Palm*, Inc. PDAs y otras licencias- 14.9% (y bajando)
- *Windows Mobile* por PDAs que cumplen las especificaciones de *Microsoft's Pocket PC* - 49.2% (creciendo)
- *RIM BlackBerry* (de *Research In Motion*) - 25.0% (creciendo)
- *Symbian OS* - 5.8% (creciendo)

- Sistemas operativos basados en el *kernel* de *Linux* para PDA's diseñadas específicamente para ellos - 0.7%
- Otros - 4.4%

2.4.3 Usos de las PDAs

Los ordenadores de bolsillo son usados para almacenar información que puede ser consultada a cualquier hora y en cualquier lugar. Por lo general, estos dispositivos son utilizados de manera doméstica, sin embargo también se pueden encontrar en otros campos:

- **En el mundo de la empresa**: cada vez es mas popular entre los directivos de las empresas dotar a sus empleados de *smartphones* que les permiten una conexión permanente a Internet y acceso al correo electrónico y gracias a los programas de sincronización de calendarios obtienen información actualizada de eventos y reuniones.
- **Usos en automóviles:** muchas PDAs son usadas en automóviles como navegadores GPS en tiempo real y es por esto que cada vez es más común encontrarlos preinstaladas en muchos coches nuevos. La popularización de estos navegadores ha sido gracias a los nuevos programas de navegación como TomTom, Garmin y iGO.
- **PDAs "robustecidas" (*ruggedized*):** desde hace años las organizaciones gubernamentales utilizan PDAs robustecidas conocidas como EDAs (*enterprise digital assistant*) para aplicaciones de datos que requieren movilidad. Algunas aplicaciones típicas son: supervisión de cadenas de producción en fábricas, entregas de paquetes, tratamientos médicos y registros en hospitales, control de accesos y seguridad, camareros de restaurantes, lectores de códigos de barras y de tarjetas, etc.
- **Usos médicos:** en la medicina las PDAs han ayudado a los diagnósticos y a la sección de medicamentos y algunos estudios han concluido que su uso por pacientes para registrar sus síntomas mejora la comunicación con los hospitales en caso de urgencia. Otros usos son acceso y actualización de bases de datos de existencias de medicamentos (*AvantGo*), bases de datos de los pacientes y sus tratamientos (*WardWacth*) y monitorización de constantes vitales de los pacientes (*Sensor Web*).
- **Usos en educación:** podemos destacar: la toma de apuntes por parte de los alumnos (que permite, entre otras cosas la corrección ortográfica automática), la distribución del material de clase por parte de los profesores a los alumnos (vía WiFi o IrDA) y como herramienta de realización de exámenes (tests).
- **Usos para personas con discapacidades:** para personas limitadas en movilidad, visión, audición y habla estos dispositivos ofrecen distintos niveles de accesibilidad, dependiendo del dispositivo en particular (incluyendo por ejemplo software de reconocimiento del habla). Recientemente, las PDAs han tenido una gran aceptación entre la población con Síndrome de Estrés Postraumático y con otro tipo de lesiones neurológicas, debido a su capacidad de almacenar eventos en el calendario, listas de tareas y que cuentan con programas recordatorios.

2.4.4 Sistemas operativos en las PDAs

2.4.4.1 Introducción

A continuación nombraremos los principales sistemas operativos que se ejecutan en la gran variedad de dispositivos PDA que pueden encontrarse en el mercado:

- **Palm OS (Garnet OS):** S.O. propietario. Fue diseñado en 1996 para su *Palm Pilot*, e implementado en una amplia gama de teléfonos móviles, relojes, lectores de códigos de barras, receptores de GPS y videoconsolas portátiles. Las últimas versiones (desde la 5.0) pueden instalarse en dispositivos con procesadores ARM. Las principales características son:
  - → Soporte de pantalla a color (hasta 480x320)
  - → Reconocimiento de escritura
  - → Reproducción de sonido
  - → Sincronización con ordenadores de sobremesa
  - → Acceso a redes TCP/IP, conectividad por puerto paralelo/USB/WiFi/Bluetooth
  - → Aplicaciones de gestión de información personal (calendario, agenda, lista de tareas, etc.) y soporte a aplicaciones de terceros.

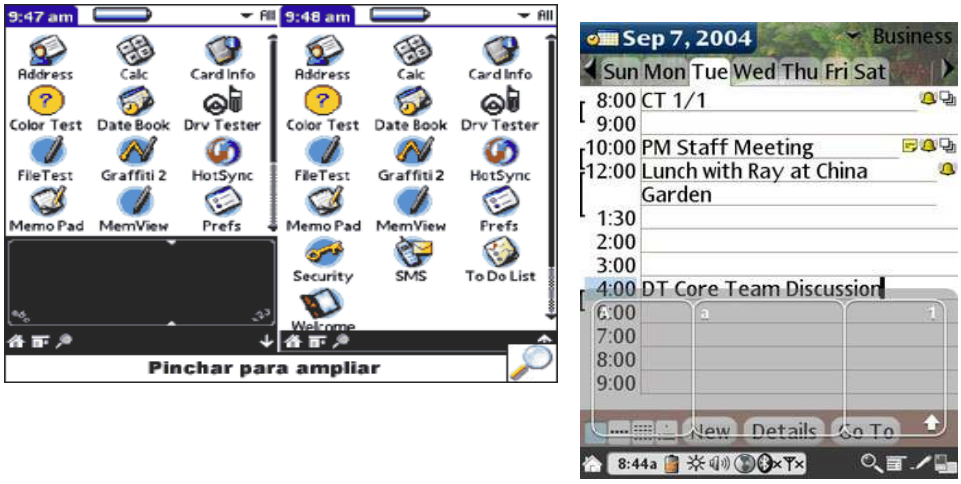


Figura 2-14: Capturas de pantalla de Palm OS 4.0 y 6.1 (cobalt)

- **Symbian OS:** S.O. propietario diseñado para teléfonos móviles, con bibliotecas e interfaces de programación (*API*) asociadas. Funciona exclusivamente en dispositivos con procesador ARM. *Symbian Ltd*. fue una asociación de *Ericsson, Nokia (en un 56.3%), Motorola* y *Psion*, creada para explotar las capacidades de los nuevos teléfonos móviles. Las características principales son:
  - → Arquitectura "microkernel" en capas: capa del kernel del SO e interfaz HW, capa de servicios del OS, capa de servicios básicos, capa de aplicaciones, capa de interfaces de usuario. Servicios de conectividad USB, Bluetooth, WiFi.
  - → Compatibilidad con medios extraíbles (tarjetas de memoria).
  - → API de gráficos y multimedia.
  - → Aplicaciones en Java ME.

- → Sincronización mediante SyncML.
- → Soporte de programación en C++ mediante SDK de Symbian (aunque también es posible en Java, VBasic y C#).
- →

Figura 2-15: Capturas de pantalla de Symbian OS

- **RIM OS:** S.O propietario multitarea exclusivo de los *smartphones BlackBerry* (1999). Estos dispositivos contaban en sus inicios con microprocesadores la familia Intel-80386 (a 80MHz) y los más modernos cuentan con procesadores ARM XScale (a 624MHz). Características:
  - → Soporte de sincronización con Microsoft Exchange Server, E-Mail y calendarios.
  - → Actualmente da soporte de conexión a redes 3G/HSDPA.
  - → Aplicaciones típicas de PDA lista de contactos, calendario, listas de tareas incluidas en el sistema operativo.
  - → Las versiones más recientes soportan pantallas táctiles y recepción de señal GPS.

Figura 2-16: Captura de pantalla de RIM OS 4.5

.

- **Familiar Linux (**http://familiar.handhelds.org/**):** es una distribución de Linux (y por tanto software abierto) inspirada en Debian, desarrollada con la idea de poder sustituir a Windows CE y Mobile en la mayoría de PDAs (de momento compatibles solo con algunos modelos de HP iPAQ). Algunas de sus características son:
  - → Distintos tipos de entornos gráficos: GPE, Opie, y bootstrap
  - → Aplicaciones típicas de PDA lista de contactos, calendario, listas de tareas incluidas en el sistema operativo en versiones Opie y GPE, además de navegadores de Internet, correo, chat, configurador de redes, etc.
  - → Disponibilidad de múltiples aplicaciones "migradas" de versiones de Linux x86.
  - → Configuración avanzada mediante consola de comandos.

Figura 2-17: Capturas de pantalla de distintas versiones de Familiar Linux

Windows Mobile: S.O. propietario, o mejor dicho, conjunto de SS.OO. basados en núcleo del sistema *Windows CE*. Actualmente existen diferentes versiones de *Windows Mobile* adecuadas a los distintos tipos de dispositivos del mercado. Ha sido diseñado para ser similar a las versiones de escritorio de Windows. Algunas de sus características son:

- Soporte para telefonía móvil *2G, 3G* y *HSDPA* (*Windows Mobile Standard* y Professional) y conectividad *Bluetooth*, *WiFi* e *IrDA* (se incluye un cliente para *RPV's PPTP*).
- Las versiones *Pocket PC* incluyen en *Windows Mobile* aplicaciones de *Microsoft Office*. Éstos incluyen *Pocket Word*, *Pocket Excel* y *Pocket PowerPoint*.
- Reproducción de archivos multimedia mediante *Windows Media Player*.
- Funcionalidad básica de las PDAs ofrecida a traces de *Outlook Mobile*. Esto incluye tareas, calendario, contactos, y la bandeja de entrada.

Figura 2-18: Capturas de pantalla de Windows Mobile 6

Hasta aquí un breve resumen de los SS.OO. que podemos encontrarnos funcionando en dispositivos PDA. Como ya hemos comentado anteriormente, en la actualidad, debido la proliferación de *Smartphones* y *PocketPC* en el mercado y a factores como la facilidad de uso y la compatibilidad con PCs de sobremesa, el sistema operativo Windows Mobile mantiene una cuota cercana al 50%. Esto hace inevitable que nos centremos a partir de ahora de forma exclusiva tanto en éste S.O. como en los dispositivos que lo soportan.

Recordemos que el objetivo de éste PFC era desarrollar una herramienta de videoseguridad que además de ofrecer una cierta funcionalidad de "ayuda" al usuario, fuese accesible desde el mayor número posible de plataformas.

2.4.4.2 Evolución de Windows Mobile

Antes de seguir con más detalles sobre este sistema operativo, haremos una breve mención del S.O del que procede: Windows CE. Éste se caracteriza por ser un S.O. compacto basado en la *API Win32* de *Microsoft* y diseñado para computadoras de mano y sistemas embebidos.

Windows CE está optimizado para su funcionamiento en dispositivos limitados en memoria: el *kernel* de Windows CE puede funcionar en menos de 1 Megabyte de memoria principal. La unidad fundamental de ejecución es el *thread* (hilo). Puede funcionar en sistemas con procesadores Intel x86 (y compatibles), MIPS, ARM e Hitachi SuperH.

La primera versión de este S.O., llamada *Pegasus* incluía una interfaz de usuario (*GUI*) con apariencia similar a la de Windows 95 y una serie de aplicaciones clásicas de Microsoft, diseñadas para su ejecución en los dispositivos de bajo almacenamiento, memoria y velocidad de la época. Desde entonces ha evolucionado hacia un sistema operativo modularizado, embebido, y de ejecución en tiempo real (con latencias mínimas). No obstante, ya no se comercializa con los nuevos dispositivos que salen al mercado.

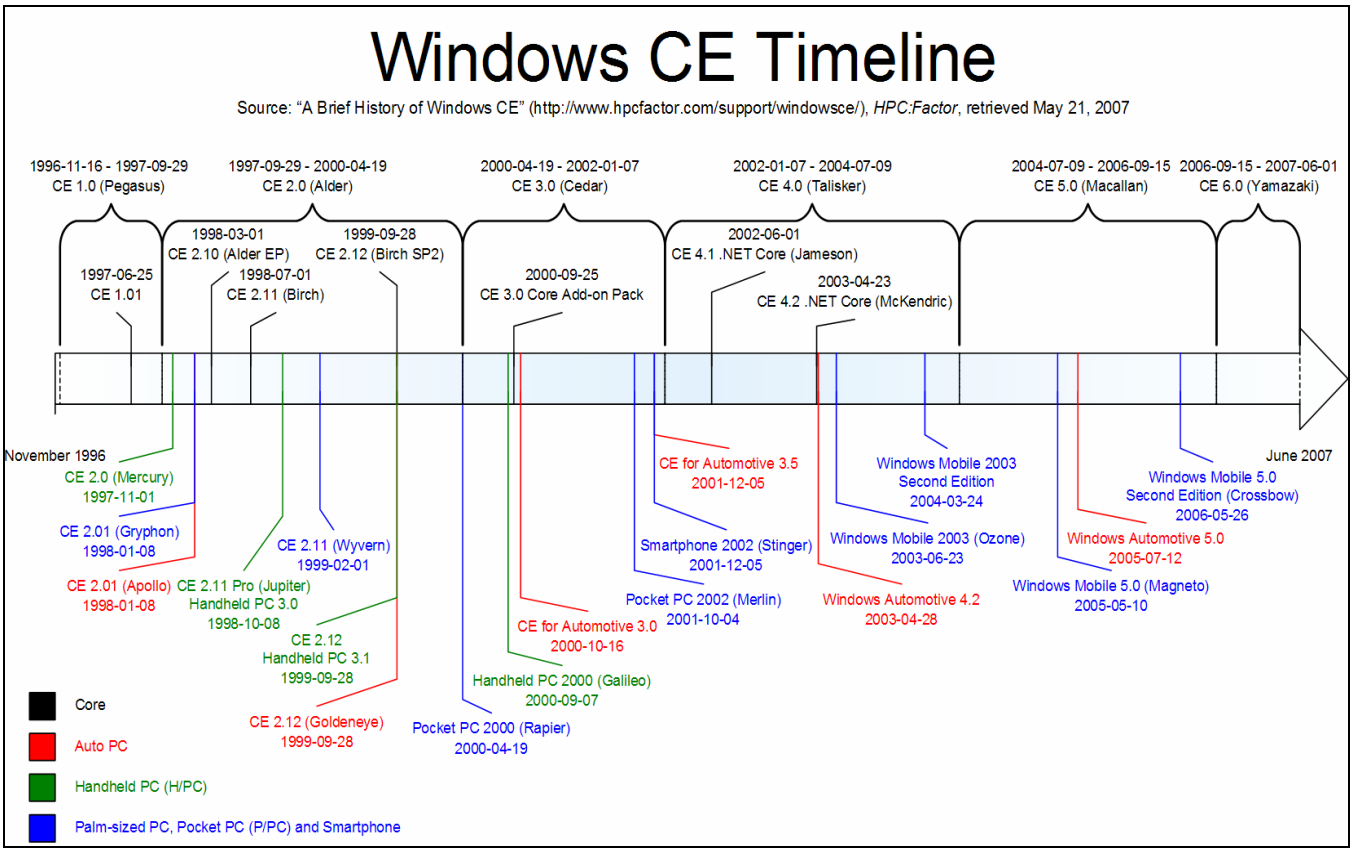


Figura 2-19: evolución de Windows Mobile y CE

Una característica importante de Windows CE en comparación a otros sistemas operativos de Microsoft es que existen partes del S.O. cuyo código está a disposición del público general (es software abierto). La idea de Microsoft era que los fabricantes utilizaran dicho código para ajustar el S.O. a sus dispositivos en particular. Sin embargo existen muchas partes del núcleo del S.O que no necesitan modificación que siguen siendo código propietario.

Muchas plataformas se han basado en el núcleo de Windows CE, algunas son: Windows PocketPC, Mobile y AutoPC. En la figura 2-19 se muestra un gráfico de la evolución de los sistemas Windows Mobile (rótulos en color azul) paralela a la de Windows CE (rótulos en color negro).

Por tanto, podríamos describir a Windows Mobile como un conjunto de plataformas basadas en el núcleo de Windows CE. En 2007, con la llegada de *Windows Mobile 6*, Microsoft abandonó el nombre *Pocket PC*, e hizo la distinción de los dispositivos en tres grupos: dispositivos Windows Mobile Classic (Pocket PC sin funcionalidades de telefonía), dispositivos Windows Mobile Standard (dispositivos con pantalla táctil) y dispositivos Windows Mobile Professional (que incluían funcionalidades de telefonía y pantalla táctil). Esas son las tres principales plataformas que llevan el nombre de Windows Mobile. Cada una utiliza diferentes componentes de Windows CE, y proporciona diferente funcionalidad para los dispositivos en los que funcionan.

La primera versión que existe es *Pocket PC 2000*. Esta versión tenía una apariencia similar a la de los sistemas operativos de la época, Windows 98 y Me, y únicamente

soportaba resoluciones de 240 x 320 (*QVGA*). En el S.O. se incluían programas como Pocket Office, Pocket Internet Explorer y Pocket Media Player. Después llegó *Pocket PC 2002*, el primer S.O. pensado también para teléfonos GSM (primeros *Smartphones*), con una apariencia similar al reciente *Windows XP*. Añadió a las funcionalidades básicas del S.O. el soporte a Redes Privadas Virtuales y a descarga de contenidos en Internet Explorer.

El primer Windows Mobile vio la luz en 2003. Supuso un gran avance en materia de conectividad, pues incluía acceso protegido a redes WiFi y Bluetooth (con diferentes perfiles de acceso), así como un cliente de L2TP/IPsec. Además podía configurarse para diferentes tipos de dispositivos (resoluciones de 640×480, 240x240, y 480x480), con posibilidad de alternar entre modo panorámico y estándar.

Windows Mobile 5 mejoró los perfiles de Bluetooth y añadió el API para el uso de receptores GPS. Un gran avance fue la inclusión del *framework* DirectShow para manejo de recursos multimedia. La siguiente versión, Windows Mobile 6 llegó en 2007 con nuevas aplicaciones y soporte para aplicaciones en red, como remote desktop, Windows Live, soporte a correo HTML, compatibilidad con JavaScript, XMLDOM y AJAX en Internet Explorer, VoIP y actualización automática del sistema operativo.

La última versión (a la espera del lanzamiento de Windows Mobile 7 a mediados de 2009) es la 6.1 es una pequeña actualización de la 6.0, con mejoras sobre todo de la interfaz gráfica de usuario (GUI).

3 Diseño

En este capítulo explicaremos con detalle como se ha diseñado el sistema de videoseguridad, así como la evolución de la arquitectura del mismo a lo largo del transcurso de este Proyecto Fin de Carrera.

3.1 Descripción de la arquitectura del sistema

En primer lugar vamos a plantear el esquema que se planteó inicialmente y que se ha seguido en todo momento para el desarrollo de este PFC. El sistema de videoseguridad distribuido se compone de dos bloques principalmente:

- Sistema inteligente multicámara.
- Sistema de streaming de video comprimido.

La figura 3-1 representa a grandes rasgos este esquema genérico:

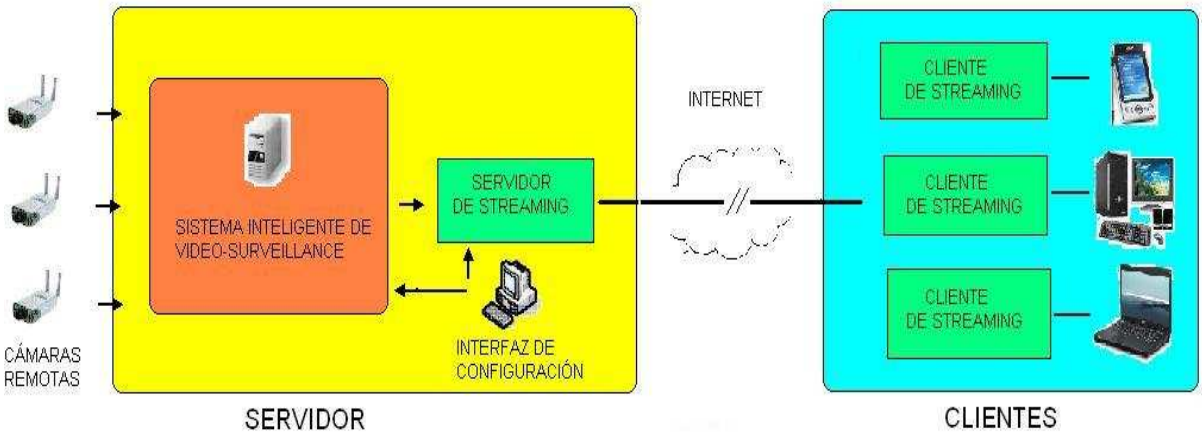


Figura 3-1: Diseño genérico de la plataforma

Pueden distinguirse ambos bloques, representados por colores naranja y verde respectivamente.

El sistema inteligente de gestión multicámara está implementado en base a la arquitectura (al framework) DiVA[8], de la que hablaremos en la siguiente subsección. En el sistema de streaming debemos hablar por separado del Servidor de streaming y del/de los cliente/s. El servidor de streaming se basa en el motor de adaptación de contenido CAIN[9]. Tanto en este capítulo como en el siguiente hablaremos de ambas arquitecturas en aspectos de diseño y desarrollo, respectivamente.

3.2 Plataforma DiVA (DIstributed Video Análisis)

Aquí explicaremos "a groso modo la funcionalidad y la arquitectura de esta plataforma que da soporte al desarrollo y ejecución de aplicaciones de análisis de secuencias de vídeo obtenidas de diferentes fuentes (cámaras, repositorios de vídeo,…), desarrollada e implementada por el Video Processing and Understanding Lab (VPULab) de la Escuela Politécnica Superior de la Universidad Autónoma de Madrid (para más información, ver [8] y [10]).

La arquitectura planteada establece un entorno distribuido para la intercomunicación simultánea de múltiples fuentes de vídeo con algoritmos de proceso, la conexión en cascada de estos algoritmos de proceso, la visualización de resultados parciales, y la inclusión formalizada de información contextual en el proceso de análisis, todo ello posibilitando que los flujos de datos se procesen en tiempo real. Actualmente esta arquitectura es la base del sistema de desarrollo de aplicaciones que el Grupo lleva a cabo en proyectos relacionados con técnicas de visión artificial.

El diseño de la arquitectura del sistema se ha basado en los siguientes criterios:

- **Escalabilidad**: la plataforma desarrollada ha ser flexible para añadir nuevos módulos de procesado (tales como nuevos algoritmos de análisis o nuevos módulos de almacenamiento de datos). También ha de soportar la adicción de nuevas fuentes de vídeo tales como cámaras de distintos tipos (USB, IP, FireWire,…)
- **Eficiencia**: la plataforma no debe añadir coste computacional a los algoritmos que operen en dicha plataforma. Además la plataforma debe permitir un procesado distribuido eficiente.
- **Generalidad**: dentro de las restricciones impuestas, la plataforma ha de desarrollarse con las herramientas y protocolos más genéricos posibles.
- **Tolerancia a fallos**: la plataforma ha de ser tolerante a fallos (tanto de los módulos principales que la componen como de los módulos de procesado de vídeo). Para ello se deben desarrollar distintas herramientas que supervisen y verifiquen el correcto funcionamiento de los componentes de la plataforma.

3.1.1 Arquitectura global

En base a dichos criterios se ha propuesto un desarrollo modular en distintos subsistemas. Cada subsistema esta relacionado con una función especifica dentro de la plataforma. Estas funciones contemplan desde la captura de datos desde distintos dispositivos físicos hasta el almacenaje y presentación de los resultados obtenidos por los algoritmos.

Los subsistemas que componen la plataforma se muestran en la siguiente figura:

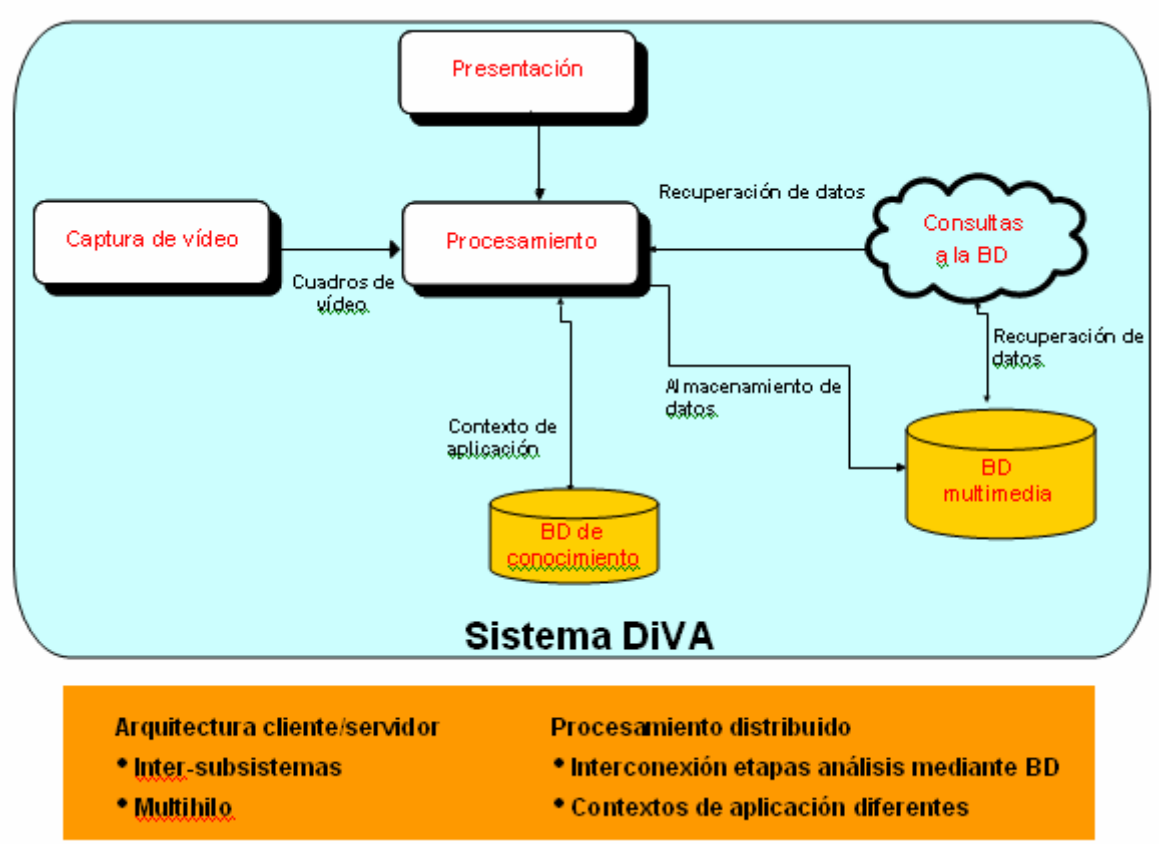


Figura 3-2: subsistemas de DiVA

El esquema de funcionamiento de cada subsistema es el siguiente.

- El **subsistema de captura de datos** reserva los recursos necesarios para su funcionamiento y comienza a capturar datos de las fuentes de vídeo que han sido seleccionadas para capturar señal de vídeo. Dicho sistema de captura dispone de un buffer para almacenar cuadros completos de la señal de vídeo que se transmitirán a los algoritmos de análisis (cuando así lo soliciten).
- El **subsistema de bases de datos** tiene dos objetivos: proporcionar un contexto de aplicación a los algoritmos de análisis y almacenar los resultados del procesado de otros módulos de análisis. Para ello dicho subsistema esta compuesto a su vez de otros dos subsistemas que proporcionan las funcionalidades anteriormente descritas.
- El **subsistema de procesamiento** tiene dos objetivos: uno es realizar el propio procesado de la señal de vídeo y otro proporcionar una interfaz de trabajo para cualquier algoritmo de procesado de vídeo (interfaz que gestiona la obtención de un contexto, el procesamiento de la imagen y el posterior almacenamiento de resultados finales o parciales).
- El **subsistema de presentación** de datos tiene por objetivo la presentación en pantalla de los datos de los análisis resultantes del subsistema de procesamiento. Este sistema permite por ejemplo visualizar en una sola ventana un grupo de imágenes con un titulo distinto.

Para compartir archivos y recursos la plataforma esta desarrollada bajo un modelo cliente/servidor en el que se separa la parte servidora de contenido (en este caso cuadros de la señal de vídeo) de la parte que lo consume (los clientes).

Los subsistemas que funcionan como servidores de contenido son los subsistemas de captura y almacenamiento de datos mientras que los subsistemas de procesamiento y presentación de datos funcionan como clientes.

A continuación en la siguiente figura se muestra un ejemplo del funcionamiento de dicha arquitectura cliente/servidor.

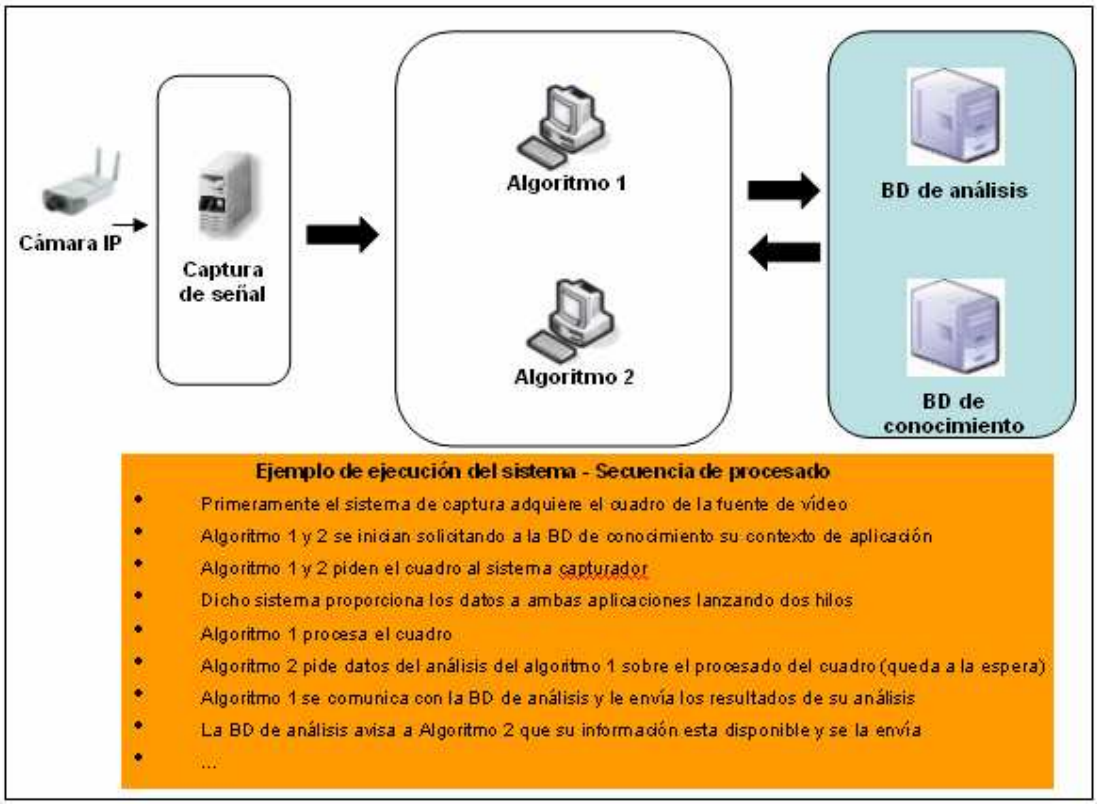


Figura 3-3: Ejecución de DiVA

3.3 Plataforma CAIN (Content Adaptation INtegration)

Esta plataforma puede describirse como un motor de adaptación de contenidos diseñado para la integración de diferentes enfoques de adaptación basado en metadatos ([10]). El principal objetivo es proporcionar un módulo de adaptación cuyas datos de entrada sean contenidos multimedia, descriptores multimedia y descriptores de contexto que permitan seleccionar los parámetros más adecuados para la adaptación y posteriormente la herramienta de adaptación de contenidos más apropiada.

El reciente desarrollo en materia de redes de acceso a Internet, en los que los proveedores de servicio ofrecen el acceso a contenidos multimedia, y el creciente abanico de terminales que se ofrecen en el mercado hacen que la adaptación de contenidos sea una cuestión clave en la oferta de futuros servicios multimedia (de acuerdo al concepto de Universal Multimedia Access (UMA), referido a la capacidad de acceso a contenidos multimedia a través de cualquier red y por medio de cualquier terminal). Además, han de tenerse en cuenta a la hora de la adaptación no solo parámetros relacionados con el terminal y la red de acceso, sino también parámetros de personalización y relativos a condiciones ambientales para así mejorar la calidad percibida por el usuario.

Los estándares de descripciones MPEG (MPEG-7 y MPEG-21) han sido seleccionados como formatos de representación de metadatos. El Módulo de Adaptación de Contenido propuesto en CAIN es un prototipo de framework para desarrollo y análisis de herramientas de adaptación de contenido.

El módulo CAIN puede ser invocado por el usuario cuando éste desee adaptar contenidos para su presentación. Adicionalmente, mientras el usuario utiliza una aplicación que le permita buscar una serie de resultados, dicha aplicación debe llamar a CAIN para pedirle adaptación de los contenidos que demande el usuario (por ejemplo, imágenes, *trailers*, etc.). Aunque el objetivo de CAIN es adaptar contenidos del modo más eficiente, se establece un compromiso entre tres factores fundamentales: coste computacional, calidad del contenido final adaptado y las limitaciones impuestas por los formatos de dichos contenidos. Más específicamente, el módulo actúa como un nodo de procesamiento encargado de:

- Seleccionar los parámetros apropiados y modalidad de funcionamiento de acuerdo a: preferencias de usuario, características de los terminales, capacidad de la red, información multimedia, variaciones y anotaciones semánticas (por ejemplo, regiones de interés).

- Seleccionar la herramienta de adaptación de contenidos CAT (*Content Adaptation Tool*).

- Configurar y manejar la CAT seleccionada para crear contenido adaptado para ser enviado al repositorio de contenido correspondiente o como servicio de streaming.

Las principales tecnologías involucradas en CAIN incluyen MPEG-7 (principalmente la especificación *Multimedia Description Schemes* –MDS[14]). MPEG-21

(principalmente la especificación *Digital Item Adaptation* –DIA[15]), y las herramientas actuales de transcodificación de contenidos y de codificación escalable que conforman los módulos de adaptación de contenidos.

3.3.1 Arquitectura de CAIN

La figura 3-4 muestra una visión general de la arquitectura de CAIN.

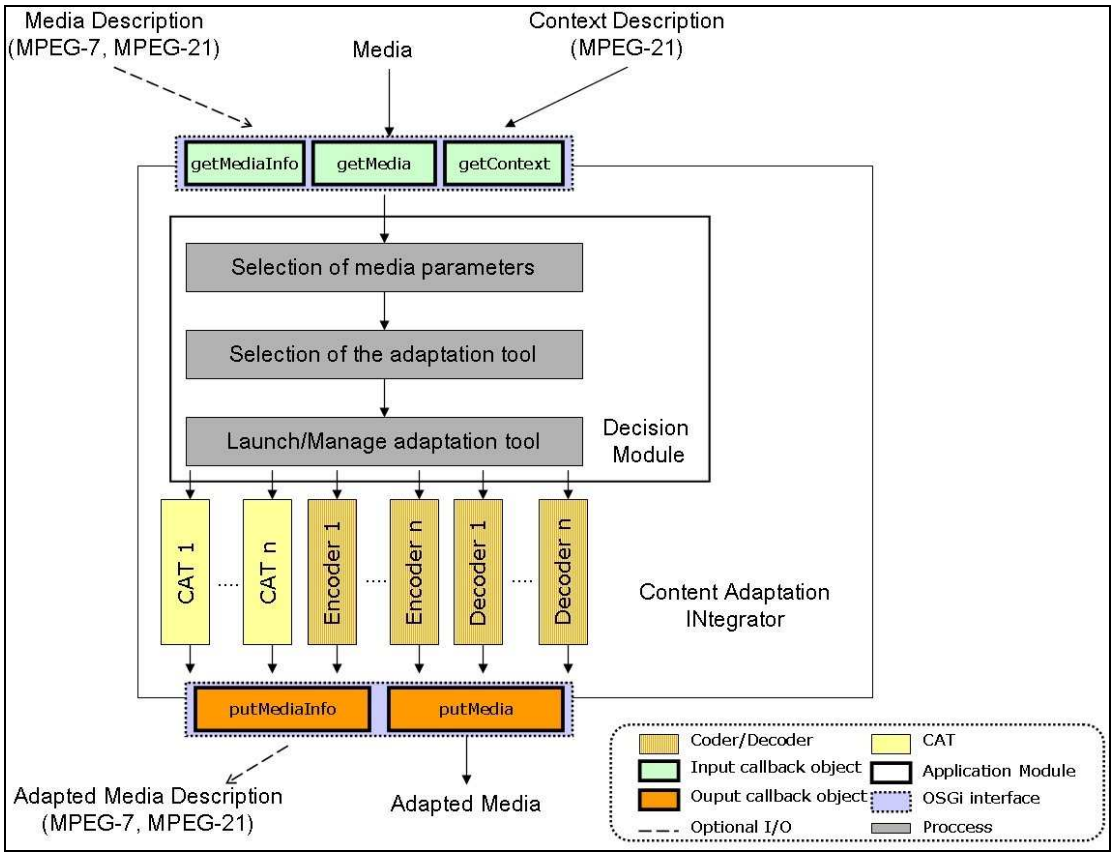


Figura 3-4: Arquitectura de CAIN

La arquitectura de CAIN está diseñada para permitir la adición de nuevos Codecs y CATs. En ambos casos es necesario desarrollar el Codec/CAT (fichero *.java, debe implementar el método `adapt(…)`) siguiendo las especificaciones de la API de CAIN y del fichero de características (fichero xml de mismo nombre que el módulo)de los CATs que incluyen información relativa a los formatos de entrada y salida aceptados por el nuevo Codec/CAT y sus capacidades de adaptación. Todos los ficheros que comprenden el Codec/CAT deben estar encapsulados en un archivo *.jar, de mismo nombre que el módulo.

En respuesta a invocaciones externas (por medio de la correspondiente API), el motor de CAIN pide y recibe contenidos, descriptores de contenidos MPEG-7 y

descriptores de *streams* MPEG-21. De forma paralela, el motor recibe descriptores de contexto compilado mediante el aceMedia User Profile que contiene, al menos, preferencias de usuario, características del terminal y capacidades de la red. Este perfil se analiza sintácticamente para poder seleccionar los parámetros de la adaptación y para decidir que CAT debe de usarse. Los CAT se ejecutan y se manejan para producir contenido adaptado y metadatos que serán transmitidos a los servicios de distribución (streaming).

Los módulos de adaptación de contenidos pueden clasificarse en diferentes categorías:

- Los módulos Transcodificadores (***Transcoders***) representan el proceso de decodificación-codificación.

- Los módulos de Contenido Escalable (***Scalable Content***) se encargan del truncado (o interpolación) de contenido escalable. Accede a flujos de datos a través de un modulo de transcodificación.

- Los módulos de Servicios en Tiempo Real (***Real Time Services***) pretenden extraer, por medio de técnicas de análisis en tiempo real, datos semánticos del contenido comprimido para generar una versión adaptada basada en semántica.

- Módulos de "***Transmoding***".

- Módulos Codificadores (***Encoders***) se encargan de codificar datos sin formato (*raw data*) según un determinado formato.

- Módulos Decodificadores (***Decoders***) se encargan de decodificar datos con un formato específico, produciendo datos sin codificar.

Para facilitar la integración de nuevos CATs en el framework de CAIN se ha implementado un mecanismo de extensibilidad que permite añadir nuevas funcionalidades de adaptación conectando dichos CATs. Además de las APIs y el mecanismo son necesarias descripciones de las capacidades de cada herramienta, que permitan a CAIN adaptar sus decisiones dinámicamente basadas en las capacidades de adaptación de los CATs recientemente añadidos.

3.3.2 Sistema CAIN

El sistema CAIN lo integran el Modulo de Decisión (DM) y el conjunto de CATs, Codecs que realizan adaptaciones, y el Módulo de Extensibilidad encargado de facilitar la inclusión de nuevos módulos de adaptación. La figura 3-5 muestra una visión general del proceso de adaptación.

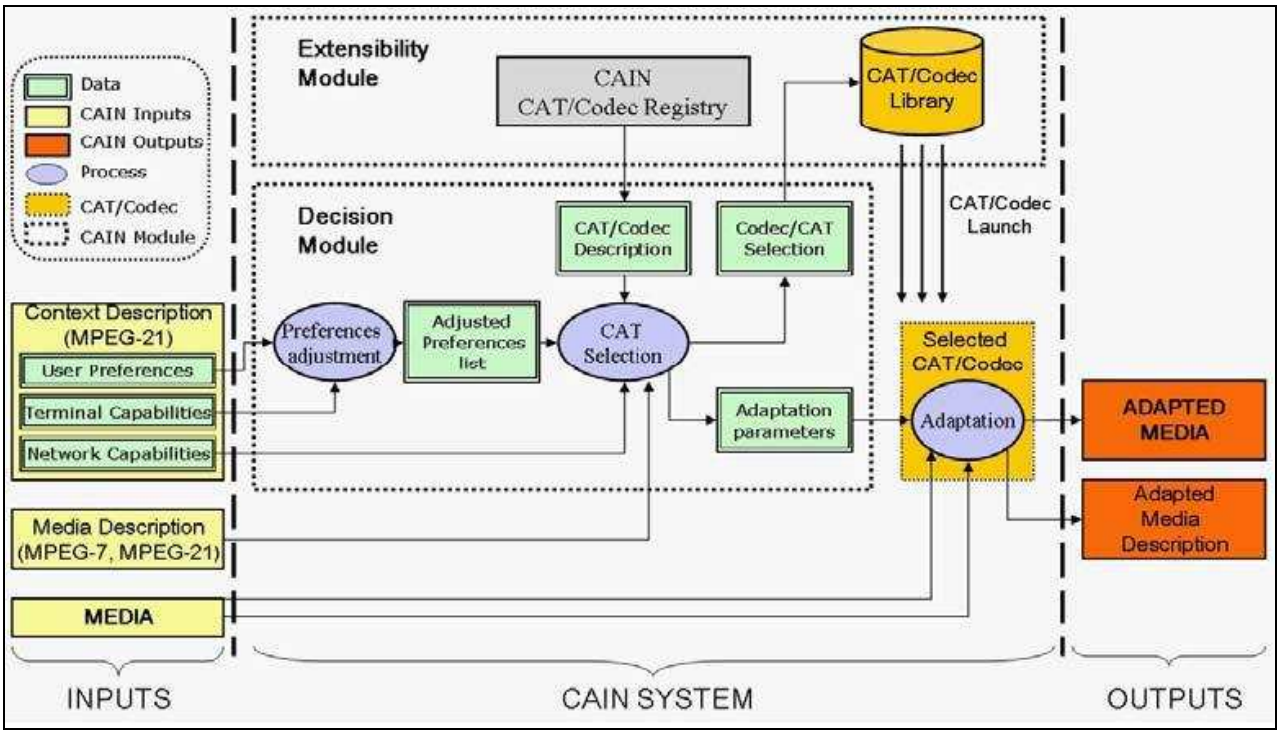


Figura 3-5: Proceso de adaptación en CAIN

- En primer lugar, las preferencias y características del terminal se contrastan para obtener una lista llamada 'Adjusted Preferences List'
- Después la selección del Codec/CAT a usar se realiza en base a dicha lista, el descriptor multimedia y las capacidades de la red. Se comprueba también las descripciones de capacidades de los Codec/CATs disponibles para obtener una lista de las herramientas que pueden desempeñar la adaptación deseada (o en caso de no encontrarlas, que desempeñen una tarea similar).
- Finalmente, la herramienta de adaptación seleccionada realiza la adaptación similar a la 'Adjusted Preferentes List', junto con el conjunto de parámetros del CAT.
- Esta herramienta es invocada para realizar la adaptación, y recibe el contenido multimedia original, el descriptor multimedia y los parámetros de adaptación definitivos. La salida del CAT es el contenido adaptado y su descriptor multimedia asociado.

El módulo de decisión (DM) se encarga de tomar estas decisiones necesarias para la adaptación. Devuelve el CAT que debe ser ejecutado y los parámetros de entrada a usar en la llamada al mismo. Este módulo puede ser considerado como la parte "inteligente" del sistema.

3.3.3 Extensibilidad en CAIN

CAIN está provisto de un mecanismo para futuras incorporaciones de nuevos CATs y Codecs. Cada nuevo CAT puede incorporarse de forma independiente, diseñando la interfaz entre el núcleo de CAIN y los módulos y proporcionando el fichero de características del CAT. Este fichero es analizado por el "Description file parser" (figura 3-6) y la información de capacidades del CAT se añade al registro "CME CAT/Codec Registry". El fichero contiene información de los formatos y parámetros que acepta el CAT, qué clase de adaptación lleva a cabo el CAT y la información necesaria para tomar decisiones acerca del uso del CAT. El módulo de decisión (DM) lee del registro las herramientas disponibles y toma decisiones en función de las características de las herramientas definidas. Las funcionalidades de extensibilidad de los Codecs también se ofrecen por medio del "*transcoding* CAT" que está desarrollado sobre *ffmpeg* [17], así como la aplicación para transcodificación de contenidos.

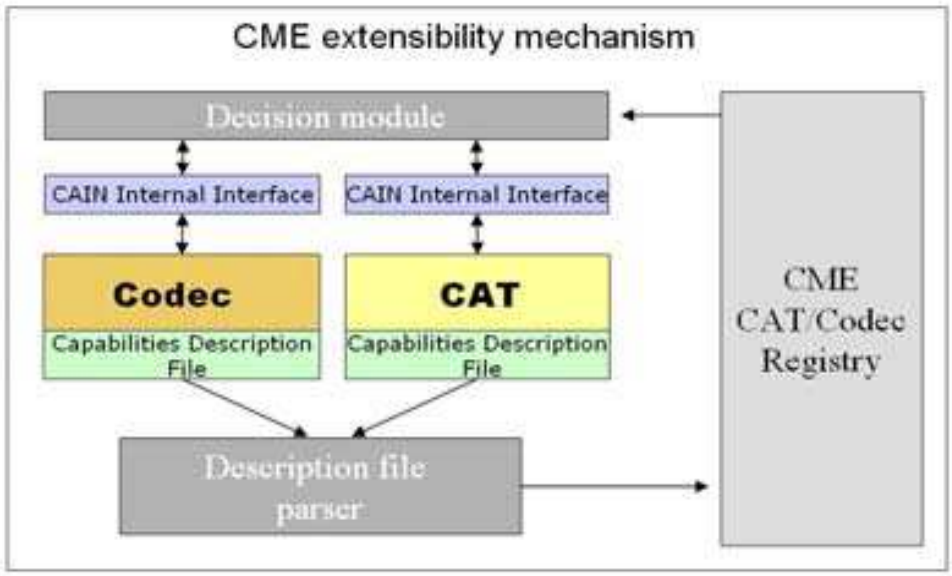


Figura 3-6: Mecanismo de extensibilidad

3.3.3.1 Interfaz de los CAT

Cuando se requiere la adaptación el DM de CAIN se encarga de seleccionar el CAT que realizará la adaptación, de seleccionar los parámetros adecuados de adaptación y de ejecutar la herramienta seleccionada. Para añadir CATs de forma genérica, es necesario definir una interfaz común. La figura 3-7 muestra un diagrama de la interfaz definida que consta de los siguientes elementos:

- Localización del contenido fuente (*Source Media Location*)
- Localización del contenido destino (*Target Media Location*)

- Parámetros de Adaptación: este argumento especifica al CAT cómo debe de adaptar el contenido fuente. Se da por sentado que si el CAT recibe una petición de adaptación del DM es porque es capaz de llevarla a cabo. De no ser así el CAT no sería seleccionado. Este argumento contiene información acerca de:

    - Modos de adaptación: es posible definir diferentes modos de adaptación en un solo CAT. Este campo especifica una modalidad de adaptación específica.
    - *Audio Coding*: especifica los parámetros de formato de codificación de audio, como el codec, la tasa de muestreo, la tasa binaria, nº de canales, etc.
    - *Video Coding*: especifica los parámetros de formato de codificación de video, como el codec, tamaño de imágenes, la tasa de cuadros, colores, etc.

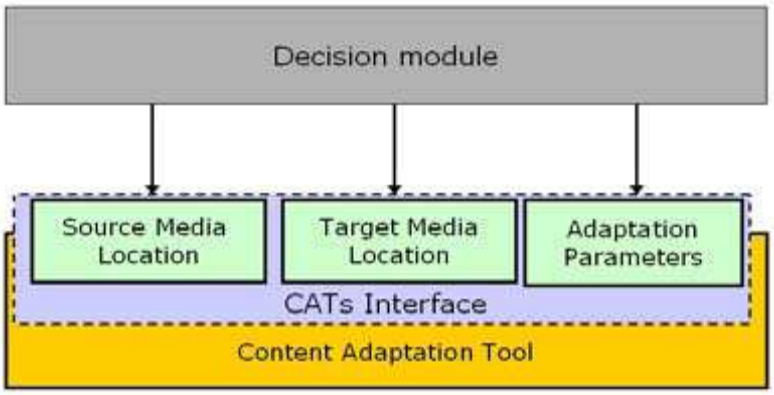


Figura 3-7: Interfaz de los CAT con el CME

4 Desarrollo

4.1 Introducción

Tras comentar brevemente las tecnologías que vamos a utilizar en nuestro diseño, explicaremos a continuación cual es el uso particular que le hemos dado a cada una de ellas. En una de ellas (DiVA) se ha hecho algún cambio en la estructura, y en la otra (CAIN) simplemente se ha realizado una implementación de una herramienta sin necesidad de alterar la arquitectura y la funcionalidad básica.

4.2 Modificaciones / Implementaciones basadas en DiVA para la creación del sistema inteligente

4.2.1 Introducción

En primer lugar, el diseño que se planteó inicialmente del sistema de videoseguridad cuando no se tenía la idea de utilizar CAIN como plataforma de streaming fue el que se muestra en la figura 4-1. La idea de utilizar las imágenes en archivos surgió de la funcionalidad de DiVA de volcado de imágenes a ficheros, que convenía aprovechar. Además, al ignorar que íbamos a utilizar CAIN para el streaming del video "procesado", preferimos funcionar con imágenes en ficheros que luego posteriormente enviaríamos a los clientes mediante un servidor FTP que enviase progresivamente las imágenes según las fuese pidiendo el cliente conectado a este. Este debería mostrar por pantalla las *frames* según fueran llegando, y con una tasa de cuadros por segundo aceptable (>15) el cliente podría así percibir movimiento continuo.

Los *frameservers* son módulos software ya implementados en la arquitectura DiVA, que más tarde comentaremos más extensamente. Los clientes son una versión modificada de algoritmos ya implementados de DiVA, llamados DiVAObjectExtractor (Clase heredada de DiVAAlgorithm). Se eligió este algoritmo porque era un cliente de cuadros que pide *frames* a su *frameserver* asociado, los procesa, obteniendo información del movimiento y los objetos de la escena y los muestra en pantalla junto con las máscaras de movimiento y de fondo (background). La modificación permitía además etiquetar estas imágenes con su nivel de importancia (prioridad) y los guardarlas en una carpeta tras todo el procesado.

El único módulo que hubo que crear de forma independiente era el selector de cuadros. Este módulo funcionaba como hilo independiente (DiVAThread) y se encargaba de recorrer carpetas con las imágenes que iban volcando los clientes de *frames*, de leer la prioridad de cada imagen en cada carpeta y volcar las imágenes prioritarias nuevamente en otra carpeta, a donde se supone que accedería el servidor de streaming para finalmente enviarlas a la aplicación de visualización remota.

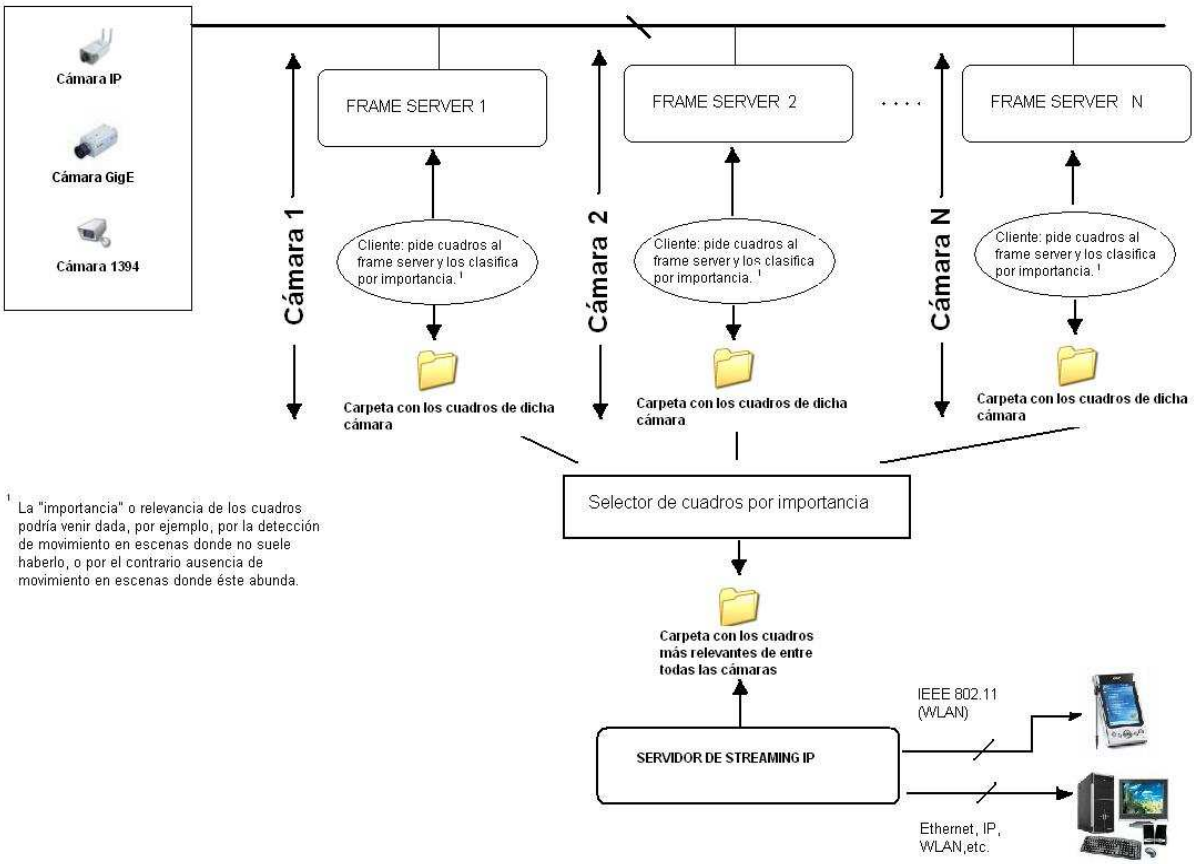


Figura 4-1: diseño inicial del sistema inteligente (DiVA)

Se suponía que al ser un programa de múltiples hilos no habría problemas de sincronización entre los algoritmos y el selector de cuadros, siempre y cuando el selector de cuadros empezase a funcionar cuando las carpetas de los algoritmos ya contuviesen imágenes (estas imágenes, al llegar a cierta cantidad serían sobrescritas nuevamente por los algoritmos, para no saturar la memoria del PC). Y de hecho así fue, eligiendo un número pequeño para el número de imágenes por carpeta (menor de 50) y configurando un módulo de renderización que mostrase por pantalla las imágenes que iba eligiendo el selector, podía observarse que la sincronización era correcta, pues en los momentos en los que no había una decisión de cambio de cámara podía verse directamente el video que grababa la cámara seleccionada.

Sin embargo, esta solución tuvo que ser rechazada, pues en las pruebas realizadas no se conseguían tasas de cuadros por segundo mayores a 5 (ni siquiera al lanzar los *frameservers* en una segunda máquina, reduciendo la carga de procesos en la primera). Por ello, hubo que pasar a la segunda y definitiva versión, en la que en vez de trabajar con las imágenes en archivos, se trabajaba con las imágenes directamente en memoria (aunque esto supuso realizar cambios en los algoritmos). Podemos ver este nuevo diseño en la figura 4-2.

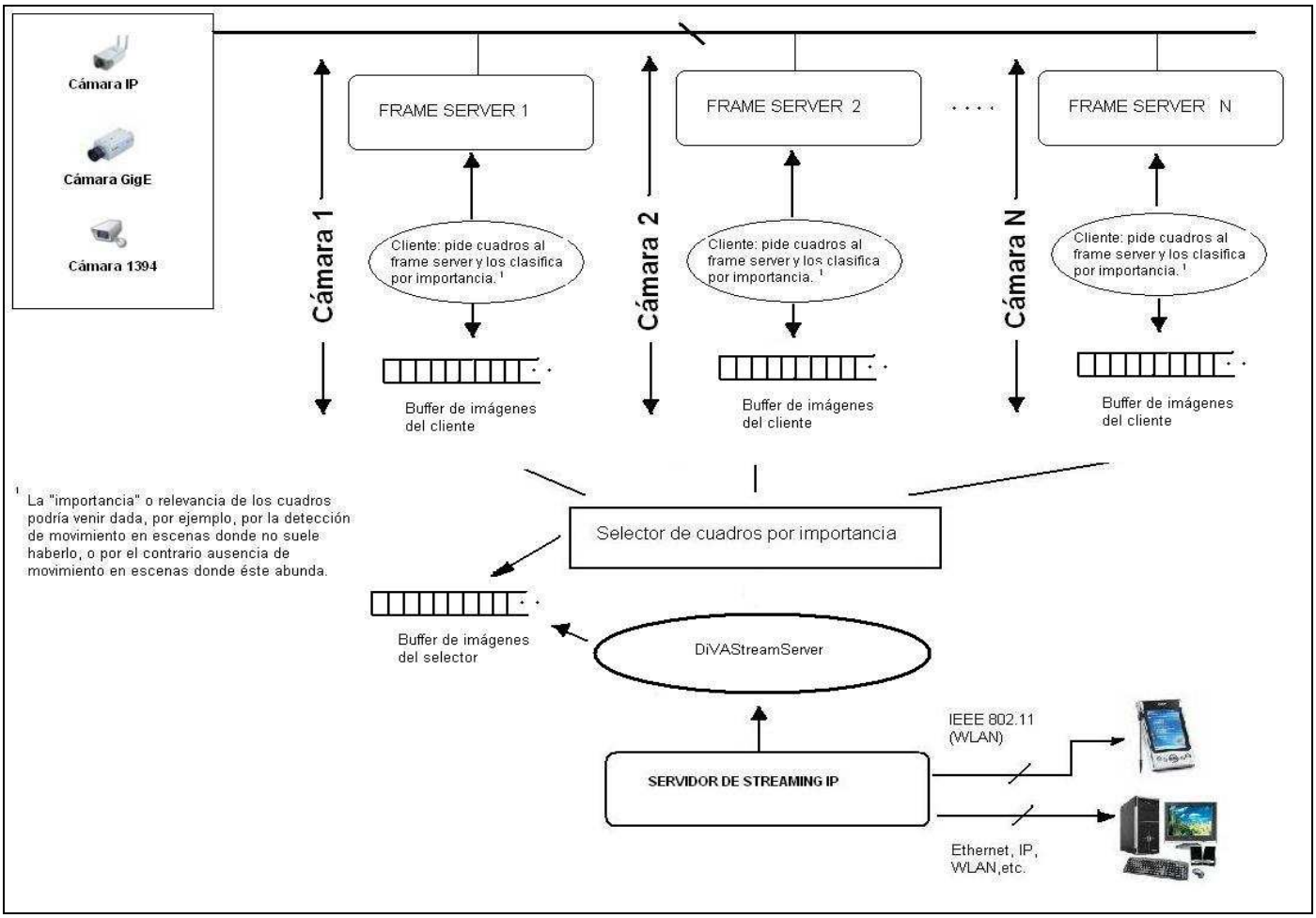


Figura 4-2: diseño actual del sistema inteligente (DiVA)

Con este diseño conseguimos aumentar la tasa de cuadros por segundo hasta 15, cuando la red interna lo permitiese (la velocidad de la conexión entre las cámaras IP y las *frameservers* limita dicha tasa).

En la siguiente subsección explicaremos con detalle la estructura de clases C++ de DiVA y los cambios e implementaciones realizadas.

4.2.2 Estructura del código de DiVA

A continuación definiremos la jerarquía de clases que componen la librería usada en la plataforma DiVA (*Distributed Vídeo Analysis*). La principal función del sistema es dar soporte a las aplicaciones de análisis de vídeo obtenido de las vídeocámaras instaladas en la Escuela Politécnica Superior de la UAM. El lenguaje de programación utilizado para generar la librería es C/C++.

La librería está organizada a dos niveles: uno básico que encapsula operaciones básicas (tales como una clase imagen, un hilo de ejecución, una conexión,…) y otras operaciones o servicios más complejos (que resultan de la combinación de las operaciones básicas).

Para realizar el desarrollo de la plataforma se han utilizado varias librerías externas. Estas librerías proporcionan diversas funcionalidades tales como manejo de conexiones TCP/IP, algoritmos de visión por computador...

Actualmente se utilizan los siguientes paquetes software:

- **Intel® Open Source Computer Vision Library versión 1.0 (OpenCV).** Esta librería se ha incorporado a la plataforma para añadir diversos algoritmos de análisis de imagen y de secuencias de vídeo.
- **Matrox Imaging Library (MIL).** Actualmente se dispone tanto de la versión 7.5 y 8.0 utilizadas con distintos fines. La versión 7.5 esta siendo utilizada para capturar información de cámaras FireWire1394 y la 8.0 para capturar datos de cámaras con el nuevo estándar GigE
- **Microsoft Fundation Classes (MFC)**. Estas librerías se usan para el manejo de conexiones TCP/IP y el uso de hilos tanto en los clientes como en los servidores de la plataforma.

Debido a las restricciones que imponen las librerías anteriormente mencionadas, la plataforma funciona bajo el sistema operativo Windows.

A continuación explicaremos brevemente algunas de las clases que componen la librería (las que involucra el nuevo código), pues más tarde cuando se comenten a fondo los cambios e implementaciones, será necesario conocerlas para entender el código y su funcionamiento.

4.2.2.1 DiVAImage

La clase DiVAImage nos sirve para representar las imágenes captadas por las cámaras como una clase de C++. La clase ha sido implementada con las funciones básicas que se le pueden aplicar a una imagen (como poner todos los píxeles a un valor, copiar una imagen,…).

Una descripción de la clase se muestra en la siguiente figura:

```
#ifndef DiVAIMAGE_H
#define DiVAIMAGE_H

#if _MSC_VER > 1000
#pragma once
#endif
/**
 *  \def DEPTH_8U
 *  Depth for pixels that are unsigned 8-bit integers
 */
#define DEPTH_8U 8

/**
 *  \def DEPTH_32F
 *  Depth for pixels that are single precision floating-point numbers
 */
#define DEPTH_32F 32

// Include need librarys
#include "cxcore.h"
#include "DiVAstructures.h"

// Class definition
class DiVAImage
{

// Indicate private attributes
private:
    ///image identification
    long idImage;
```

```
    ///image timestamp
    long timestamp;
    ///image
    IplImage *image;


// Show public method
public:

    ///Default constructor
    DiVAImage();
    ///Constructor with basic parameters
    DiVAImage(long sizeX, long sizeY, int nChannel ,int Type);
    ///Constructor with a image in disk
    DiVAImage(char *filename,int iscolor=1);

    ///Class's Destructor
    virtual ~DiVAImage();

    ///Method to access to image pixels
    DiVASCALAR getPixel(int x, int y);

    ///Method to set one image pixel to a value
    void setPixel(int x, int y, DiVASCALAR valor);

    ///Set all pixels in the image to a reference value
    void setPixels(DiVASCALAR value);
    ///Set all pixels in the image to a reference matrix
    void setPixels(void *value);
    ///Gives direct access to pixels in the image
    void *getPixels();

    ///Set identification number
    void setId(long id);
```

```
    ///Set time stamp
    void setTimeStamp( long timeStamp);
    ///Set origin of pixels data
    void setVAlign(int value);
    ///Get the indentification number
    long getId();
    ///Get the time stamp
    int getTimeStamp();
    ///Get image width in pixels
    int getWidth();
    ///Get image height in pixels
    int getHeight();
    ///Get number of channels
    int getNChannels();
    ///Get type of image data
    int getDepth();
    ///Get image color model (RGB,HSV,...)
    char *getColorModel();
    ///Get origin of pixels data
    int getVAlign();
    ///Get Image data size in bytes
    int getDataSize();
    ///Flips image
    void flip(int flip_mode);
    ///Method to save a image in disk
    int saveImage(char *filename);
    ///Method to load a image from disk
    int loadImage(char *filename,int iscolor=1);
    ///Method to obtain a equal copy of the actual image
    DiVAImage *clone();
    ///Method to set the image pixels with the pixels content for the input image
    int copy(DiVAImage *image);
    ///Converts the image from RGB to Gray
    int RGB2gray();
};

#endif
```

Figura 4-3: Clase DIVAImage

La arquitectura de la clase DiVAImage gira en torno a la estructura IplImage de OpenCV. Sobre la IplImage se construyen diversas funciones para el funcionamiento de una imagen (tales como cargar/guardar ficheros a disco, acceso eficiente a los píxeles,…).

Cada imagen recibe una idImage única (asignada por la fuente capturadora de cuadros (imágenes) y un timeStamp que indica su instante de captura.

4.2.2.2 DiVAThread

Para el manejo de hilos en nuestra arquitectura utilizamos la clase DiVAThread. Dicha clase está basada en multihilo mediante MFC (Microsoft Fundation Classes). La estructura de la clase es la siguiente:

```
#ifndef DiVADTHREAD_H
#define DiVADTHREAD_H

class DiVAThread
{
    public:
    DiVAThread();
    ~DiVAThread();
    int ExecutionLoop();
    int start();
    int stop();
    int isActive();
    virtual int initResources(){return 0;};
    virtual int releaseResources(){return 0;};
    virtual  int process(){return 0;};
    int isEnd(){return _isEnd;};

    ///Method to set the thread name
    char* setName(char *name);
    ///Method to obtain the thread name
    char *getName(){return _threadName;};
    protected:
    virtual int setEnd(){_isEnd=1;return _isEnd;};
    private:
    int _isEnd;
    char _threadName[64];
    void*  _pExecutionThread;
};
#endif
```

Figura 4-4: Clase DiVAThread

Dicha clase nos permite lo siguiente:

- Reserva inicial de recursos (método initResources( ))
- Ejecución continua (método process())
- Liberación de recursos (método releaseResources( ))

Para que el hilo que se pretende crear comience su funcionamiento se tiene que llamar al método Start( ) de la clase. Para finalizar la ejecución del hilo, simplemente con llamar al método Stop( ) bastará.

En la arquitectura se utiliza la clase DiVAThread para añadir procesamiento multihilo a las distintas aplicaciones que forman parte de la plataforma DiVA. Así pues si queremos que el sistema capturador (*frameserver*) comience a servir cuadros a los usuarios y a la vez realice una captura continua, se plantean dos posibles modos de funcionamiento: secuencial y multihilo.

Si elegimos el modo multihilo, la clase que implemente la captura de cuadros de la fuente heredará de la clase DiVAThread y así mediante el método Start( ) se realiza una ejecución iterativa del método process( ) (en dicho método es donde está instanciada la captura).

4.2.2.3 DiVACapture

Para cada fuente de vídeo, se necesita una capturadora que haga de interfaz entre el software del dispositivo fuente y la plataforma DiVA. La clase DiVACapture nos permite capturar cuadros (frames) desde una fuente de vídeo (o de imágenes en JPEG) y las va almacenando en un buffer.

La clase DiVACapture es abstracta y se deriva una clase hija de ella por cada tipo de fuente de la que se pueden capturar los datos.

La capturadora soporta dos modos de funcionamiento: bajo demanda y continuo. En el modo bajo demanda es el usuario el que indica la captura (o el sistema de captura automáticamente) y en el modo continuo se realiza una captura continua (lanzando un hilo paralelo que realiza dicha tarea). En el caso del modo continuo, el sistema de captura examina el buffer interno de la capturadora y hace disponible dicho recurso para que lo utilice el servidor de cuadros del sistema de captura.

Dicha clase tiene un buffer interno en el que va almacenando todas las imágenes leídas. A continuación se muestra el fichero de cabecera de la clase Capturadora

```
#if !defined(AFX_DiVACAPTURE_H)
#define AFX_DiVACAPTURE_H

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

#define DEFAULT_BUFFERLENGTH 5 //Frames
#include "DiVAImage.h"
#include "DiVAThread.h"
#include "DiVAImageBuffer.h"
#include "DiVACrono.h"

class DiVACapture:public DiVAThread
{
public:

    ///Constructor
    DiVACapture(int bufferLength=DEFAULT_BUFFERLENGTH);
    ///Destructor
    ~DiVACapture();
    virtual char* getSourceId(){return NULL;};
    int process();
    int start();

    ///Inicialization
    virtual int init();
    /// Get current frame from source
    virtual int getCurrentFrame(DiVAImage* pImage);
    /// Get the next frame from source and advance to it
    virtual int getNextFrame(DiVAImage *nextFrame);
    /// Only capture next frame and no image is returned
    virtual int DiVACapture::captureNextFrame();
    DiVAImage*  getSampleFrame(){return _frame;};
```

```
    DiVAImageBuffer*        getImageBuffer(){return _pimageBuffer;};
    int writeInBuffer(DiVAImage* pImage);

    int setBufferLength(int bufferLength){_bufferLength=bufferLength; return _bufferLength;};
    int setConsumeControl(int consumeControl){_consumeControl = consumeControl;return _consumeControl;};
    int getConsumeControl(){return _consumeControl;};
protected:
    DiVAImage *_frame;
    DiVAImageBuffer*    _pimageBuffer;
    int             _bufferLength;
private:
    DiVACrono*      _pCrono;
    int _consumeControl;
    ///Image stored by the capture

};

#endif // !defined(AFX_DiVACAPTURE_H)
```

Figura 4-5: clase DiVACapture

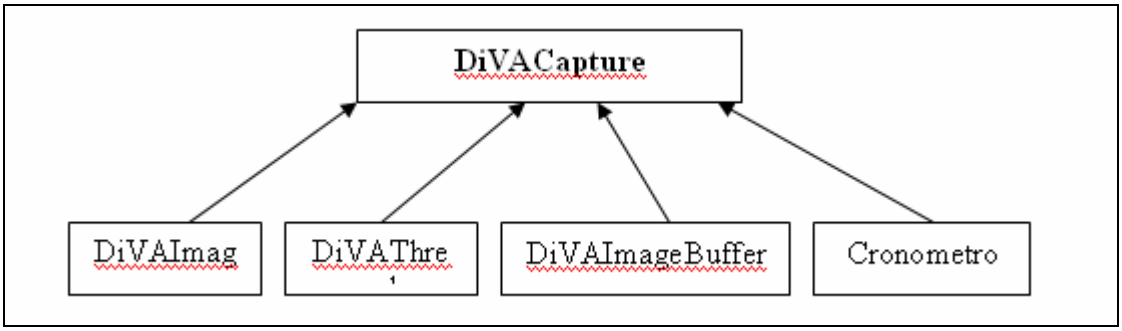


Figura 4-6: jerarquía de clases para DiVACapture

4.2.2.4 DiVAClient

Para poder permitir la conexión de los algoritmos y distintos subsistemas que usan el modelo cliente/servidor con aquellos sistemas que ofrecen un servicio (como el servidor de frames o el servidor de datos), se necesita de la clase DiVAClient. Dicha clase proporciona las funciones necesarias TCP/IP para poder realizar la conexión y transmisión de datos.

```
#define BUFFERSIZE = 65000;//3*800*600

class DiVAClient
{
public:
    DiVAClient(char* servername, int port=20248);
    ~DiVAClient();

private:
    ///IP server name
    char*   _servername;
    ///Connetion port
    void*   _pconn;
    ///TCP/IP settings
    struct hostent *_hp;
    int _isEnd;
    int     _port;
    int     _idClient;
    char*   _sourceId;
    int SendCommand(char* command);
    int WaitForAnswer(void**);
    int WaitForACK();
    int WaitForId();
    int DiVArecv(char*buffer ,int longitud);

public:
    ///Method to connet with server
    int Connect();
    ///Method to known if task is end.
    int isEnd(){return _isEnd;};
    ///Method to set END status.
    int setEnd();
    ///Method to receive a frame from server
    int receiveFrame(void** pbuffer,long idFrame);
    ///Method to set client ID
    int setId(int idClient){_idClient=idClient;return _idClient;};
    ///Method to get client ID
    int getId(){return _idClient;};
    ///Get additional information about frameSource
    char* getSourceId(){return _sourceId;};
    ///Set additional information about frameSource
    char* setSourceId(char* sourceId);
    //Method to test basic operation
    static int testFrameClient(char* servername);
};
```

Figura 4-7: clase DiVAClient

4.2.2.5 DiVAFreeBuffer

Para poder almacenar y permitir un acceso eficiente de datos (como son los cuadros que maneja la plataforma) como por ejemplo guardar los frames recibidos (por el cliente), capturados (por el DiVACapture) o para cualquier otro propósito de almacenamiento, se define la clase DiVAFreeBuffer.

Esta clase es abstracta y solo define algunos métodos generales que las distintas clases derivadas tendrán que implementar. Este buffer permite un control de datos enviados y de usuarios registrados.

Implementaciones de esta clase abstracta son DiVAImageBuffer.h (para almacenar DiVAImage), BufferMil (para trabajar con las imágenes de las cámaras 1394) y DiVASharedBuffer (buffer con control de usuarios y distintas políticas de vaciado)

4.2.2.6 DiVAServer

Para poder proporcionar un servicio de distribución de frames a través de una red TCP/IP se necesita una clase que realice dicha tarea.

Para implementar dicho modulo se han utilizado las funciones de red de Windows. La tarea principal del servidor es quedarse en modo espera y atender las peticiones de los clientes lanzando hilos que atiendan el servicio.

```
class DiVAServer:public DiVAThread
{
public:
    DiVAServer(void* source,int portNumber);
    ~DiVAServer();
    //Threading control
    int stop();
    int process();
    int initResources();
    int releaseResources();
    int processCommands(char *str,void* parametros);
    int recvCommand(void* pclient,char*buffer);
private:
     void*      _source;
     int        _portNumber;
     void*      _pserver;
     int        _end;
     void*      _clientTable;
     int getPortNumber(){return _portNumber;};
    DiVAImage* getFrame(long idFrame, long idConsumidor, void* source=NULL);
    int     getId(void* source=NULL);
    int     releaseId(long idClient,void* source=NULL);
    void*       setServerSocket(void* pserver);
    void*       getServerSocket(){return _pserver;};
    int         addClient(void* pclient);
    int         releaseClient(int _idclient);
    int         releaseAllClients();

public:
    static int testFrameServer(char* filename);

};

typedef struct
{
    int id;
    void*       pclientsocket;
    DiVAServer* padre;
} PARAMETERS;
```

Figura 4-8: Clase DiVAServer

4.2.2.7 DiVAAlgorithm

La clase DiVAAlgorithm nos permite definir de manera rápida y eficiente un algoritmo en la plataforma DiVA. Dentro de dicha clase se encuentran todas las funciones necesarias para su funcionamiento. A continuación se muestra la arquitectura de clases que componen DiVAAlgorithm.

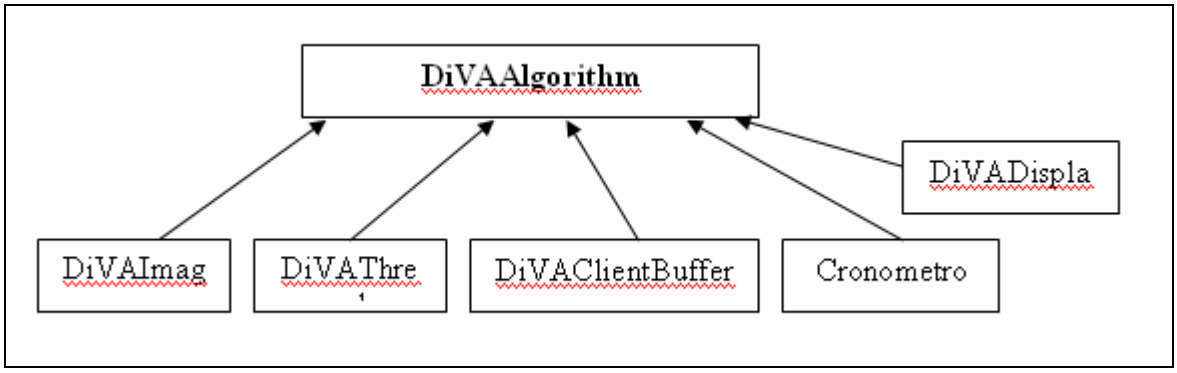


Figura 4-9: Arquitectura de DiVAAlgorithm

```
class DiVAAlgorithm:public DiVAThread
{
    public:
    DiVAAlgorithm(char* frameServername, int portnumber,
            BOOL fileDumping=TRUE,
            BOOL display=TRUE,
            char* dataServerName=NULL,
            char* contentServerName=NULL,
            BOOL dataServerDumping=FALSE,
            BOOL contentServerDumping=FALSE,
            int captureMode=CAPTURE_BUF);
    ~DiVAAlgorithm();
    char* getAlgorithmName(){return _algorithmName;};
    //Initialization
    virtual int init();
    //Execution Control
    int setEnd();
    int process();

    //Stop algorithm
    int stop();

    //Connections
    int connect();
    int getFrame(DiVAImage** pImage,long idFrame);
    int receiveFrame(DiVAImage** pImage,long idFrame);
    /*************************************************************************************************/
    /*To be overwriten*******************************************************************************/
    //Data bases
    virtual int receiveData(void** data){return 0;};
    virtual int receiveContentData(void** data){return 0;};
    virtual int releaseData(void* data){return 0;};
    virtual int releaseContentData(void* data){return 0;};
    virtual int dumpData(){return 0;};
    virtual int dumpContentData(){return 0;};
    virtual int dumpResultsInFiles(){return 0;};
    //Image Processing
    virtual int processFrame(DiVAImage* pImage,void* pdata=NULL,void* pContentData=NULL);
    //Displaying
    virtual int refreshDisplay(){return 0;};
```

Figura 4-10: Clase DiVAAlgorithm

Debido a que la clase DiVAAlgorithm hereda directamente de DiVAThread, al iniciar la ejecución de nuestro algoritmo, la clase DiVAThread se encarga de ejecutar periódicamente un método que llama al método processFrame(…) de la clase DiVAAlgorithm. El análisis de imagen/vídeo entonces tendrá que encontrarse en el método processFrame(…) de la clase DiVAAlgorithm.

Los resultados son mostrados en el método refreshDisplay( ) que es llamado justo después del procesado del método processFrame(…).

4.2.3 Arquitectura del Sistema Inteligente de videoseguridad

Como ya vimos en la sección 4.2.1, inicialmente partimos de un esquema general como el de la figura 4-1 que resultó no ser válido por su alto coste computacional. Posteriormente evolucionó a otro que contemplaba su integración junto con la plataforma CAIN para el streaming del video comprimido. La siguiente figura 4-11 muestra el diseño final desde el punto de vista de código (las clases de la librería de DiVA involucradas).

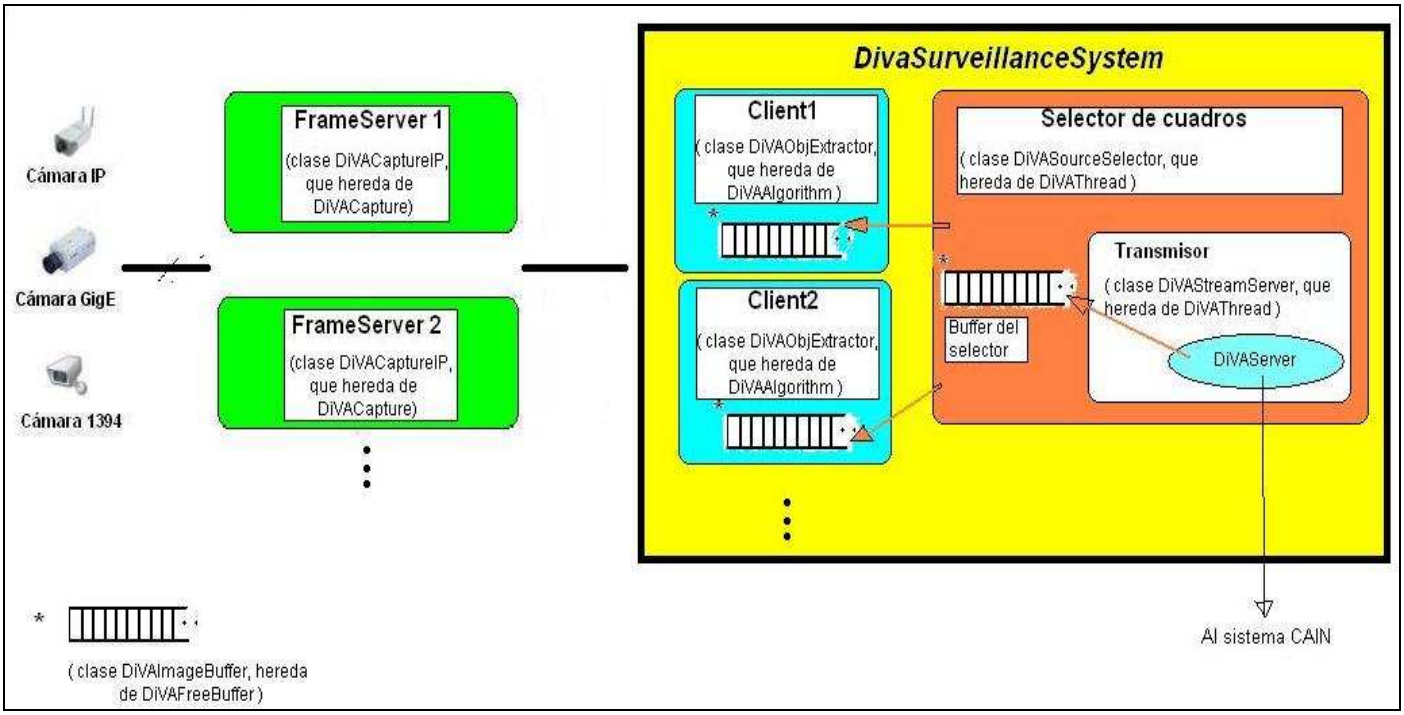


Figura 4-11: arquitectura del sistema inteligente de videoseguridad

En la siguientes subsecciones explicaremos con detalle el funcionamiento de cada uno de los bloques que componen este sistema.

4.2.3.1 FrameServers (servidores de imágenes)

El sistema de captura de *DiVA (o FrameServer)* consta de tres módulos:

- **Un modulo capturador de datos**: dicho módulo se encarga de ir proporcionando *frames* al *FrameServer*.
- **Un buffer para el almacenamiento de los datos**: dicho módulo se encarga de almacenar los *frames* proporcionados por el capturador y se los proporcionara al servidor de *frames* cuando este tenga que servir a un cliente.
- **Un servidor de frames:** dicho módulo se encarga de enviar los *frames* almacenados en el buffer por red TCP/IP a los clientes que soliciten dichos *frames*.

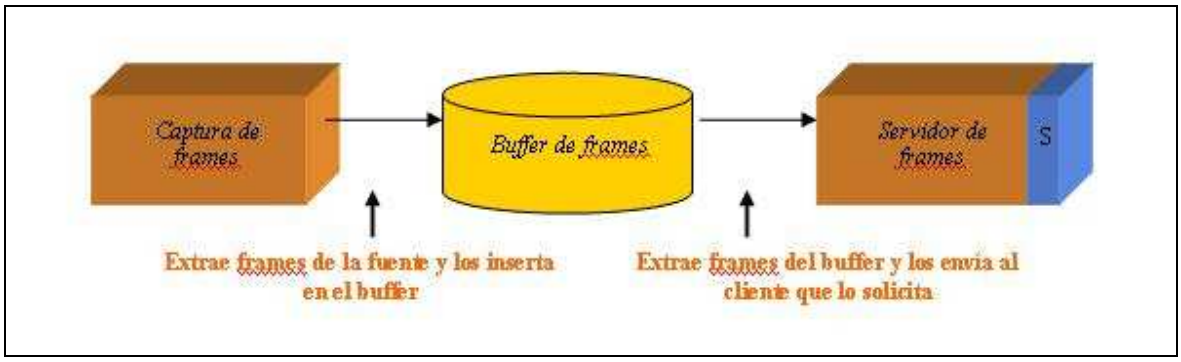


Figura 4-12: Estructura del *frameserver*

La secuencia de funcionamiento del *FrameServer* es la siguiente:

1. **Arranca el *FrameServer*** con los parámetros de configuración correspondientes (extraídos del fichero de configuración o por defecto)
    a. Se inicializa el buffer de *frames*
2. **El *FrameServer* lanza un hilo** paralelo a su ejecución **para capturar los frames**. Las funciones de este hilo son:
    a. Inicializar el capturador de *frames*
    b. Establecer los parámetros de configuración del capturador (tales como tamaño de la captura, frecuencia (fps), formato (RGB24,…))
    c. Comenzar la captura de frames

i. La captura de *frames* se ejecuta en un bucle infinito que se realiza periódicamente (cada fps).
ii. La única condición de terminación de la captura es que el *FrameServer* sea parado. En el caso de que una captura no sea posible, esta se obvia y se continúa con el proceso de obtención de frames periódico.

3. **El *FrameServer* lanza otro hilo para realizar el servicio de distribución de *frames*.**

   a. Se establecen los parámetros básicos para el servicio distribuido de *frames* a toda la red. Posteriormente se pasa al modo de escucha de petición de *frames*.
   b. Por cada petición recibida, se lanza un hilo que sirve dicha petición. Este hilo busca el *frame* en el buffer y lo envía al cliente si el *frame* existe

4. Posteriormente el ***FrameServer* se queda bloqueado** esperando a terminar. Durante este bloqueo el servidor de frames se quedan esperando peticiones de cuadros de otros módulos de la plataforma Dicha terminación solo se produce por intervención directa del usuario y se realiza cuando el usuario pulsa la tecla ESC.

Gráficamente el esquema de ejecución es el siguiente:

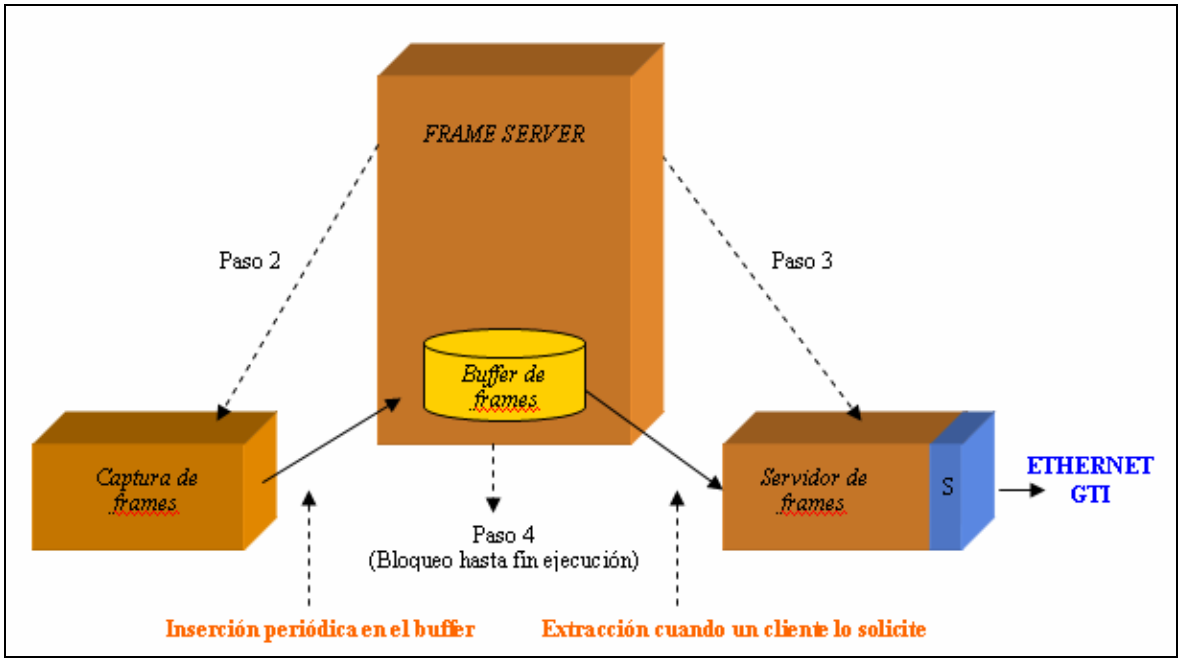


Figura 4-13: Funcionamiento del frameserver

La captura de *frames* tiene varios parámetros de configuración (la existencia o no de todos depende del tipo de capturadora que se use). Estos son:

- *Tamaño de los frames*, este parámetro nos indica el tamaño de la imagen capturada (320x240, 640x480, 1024x768,…)
- *La frecuencia de captación de frames (fps)*, este parámetro nos indica cada cuanto tiempo el hilo que ejecuta la captura debe escribir datos en el buffer del *FrameServer*. Este parámetro debe existir para todas las capturadotas que se implementen.
- *Formato de imagen capturada*. Si la capturadora lo soporta, debe ser configurable el tamaño del *frame* capturado y el formato (RGB-24, Escala de Grises 8Bpp,…)

NOTA: la capturadora deberá asignar un ID único a cada *frame* capturado de la fuente. Este número nos sirve para una identificación única de la imagen en toda la plataforma *DiVA*.

El buffer de *frames* nos permite almacenar un cierto número de cuadros procedente de la capturadora. A grandes rasgos, se permiten dos modos de funcionamiento: uno sin control de usuarios y otro con control.

- En el modo sin control de usuarios, cada usuario pide al *FrameServer* el *frame* que desea (identificado por un ID asignado por la capturadora) y este le proporciona ese *frame* (o en el caso de no encontrarlo el *frame* con el ID más próximo (por exceso y nunca por defecto)). En este modo de funcionamiento, es el propio algoritmo el que lleva el control de los datos que procesa.
- En el modo con control de usuarios, cada usuario tiene una ID y se lleva un control de los datos que piden los usuarios y los que son servidos por el *FrameServer*. Este control nos permite que los distintos algoritmos se puedan sincronizar y trabajar

colaborativamente, distribuidamente,… En este modo, el *FrameServer* lleva un control de los datos que son enviados a cada algoritmo para su proceso.

El servidor de *frames*, se encarga de enviar los *frames* solicitados a los distintos clientes del sistema. Para realizar esta tarea, se basa en una arquitectura cliente-servidor. Se usa un modelo *multihilo* que permite servir contenido a varios usuarios a la vez. El protocolo de comunicación utilizado es TCP/IP.

El Servidor de *frames* consta de dos parámetros para su configuración:

- Un puntero al buffer de imágenes
- Un puerto por el que se escuchan las peticiones.

Después de ser configurado correctamente, el servidor se pone en modo espera para ir sirviendo a los distintos clientes que solicitan datos. Para cada solicitud de datos de un cliente, el servidor lanza un hilo que atiende la petición y le sirve el contenido. Posteriormente dicho hilo finaliza.

Existen diferentes "clases" que heredan de DiVAClient para cada una de las fuentes de imágenes de las que se recogen los *frames*. Estos son:

- **DiVACaptureAVI**: Este capturador nos permitirá obtener frames de un video en disco. El formato de ese video es AVI (vídeo sin comprimir). Para iniciar dicha capturadora lo único que hace falta es la ruta del fichero en disco.
- **DiVACaptureUSBCam:** Este capturador nos permitirá obtener frames de una cámara USB conectada al PC. En el caso de que existan varias cámaras conectadas a un PC, existe la posibilidad de que se seleccione de que cámara obtener los *frames*.
- **DiVACapture1394**: Este capturador nos permitirá obtener datos a través las cámaras FireWire1394 instaladas en la Escuela Politécnica Superior.
- **DiVACaptureIP:** Este capturador nos permitirá obtener datos a través las cámaras PTZ instaladas en la Escuela Politécnica Superior. Dichas cámaras utilizan la red Ethernet para realizar el envío de los datos capturados.
- **DiVACaptureGigE**: Este capturador nos permitirá obtener datos a través las cámaras de alta definición que utilizan el interfaz GigE.

Aunque el diseño que hemos realizado serviría con cualquiera de estas capturadoras, ha sido probado con *frameservers* del tipo DiVACaptureIP (explicaremos esto en el siguiente capítulo).

A continuación explicaremos el funcionamiento del buffer compartido implementado en la plataforma *DiVA*. El buffer compartido nos servirá para almacenar los frames que nos proporcione la fuente (cámara IP, 1394, USB,…) y posteriormente servir dichos datos a los usuarios que los soliciten.

El Buffer de *frames* se utiliza principalmente en el FrameServer para almacenar los *frames* que son obtenidos por los capturadores del *FrameServer*.

En la siguiente figura se muestra la localización del *FrameBuffer* (en el F*rameServer*)

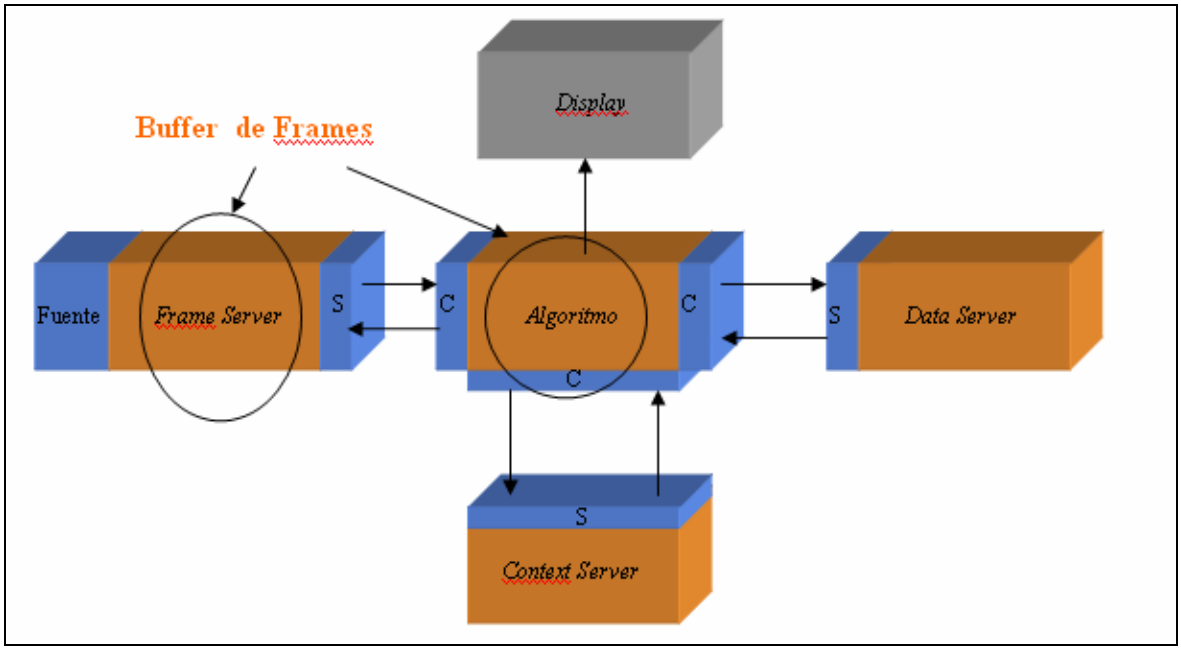


Figura 4-14: Uso de buffers de cuadros en la plataforma *DiVA*

En nuestro proyecto el buffer de cuadros es del tipo DiVAImageBuffer, que hereda de DiVAFreeBuffer (buffer genérico, ver 4.2.2.5). Esto nos permite utilizar directamente imágenes del tipo DiVAImage. DiVAImageBuffer encapsula la gestión de datos y consumidores registrados para el correcto funcionamiento del buffer.

Ya hemos visto el funcionamiento de los servidores de cuadros, y a continuación en la siguiente subsección explicaremos de qué manera acceden los clientes a las imágenes que van capturando a través del buffer compartido que mencionamos anteriormente).

4.2.3.2 Clientes de imágenes

Como ya vimos en la figura 4-11, el programa DiVASurveillanceSystem contenía varios “clientes” que conectaban con los *frameservers* con el fin de obtener las imágenes capturadas por estos y posteriormente procesarlas. Estos clientes, desde el punto de vista de la arquitectura DiVA, son DiVAAlgorithm’s (algoritmos de procesado). En este proyecto, se han elegido los algoritmos ObjExtractor (extractor de objetos) pues nos daba la oportunidad de obtener información de movimiento en la escena y de los objetos detectados en la misma, con un coste computacional aceptable. A continuación explicaremos con detalle el funcionamiento de este algoritmo.

En la siguiente figura 4-15 podemos observar la arquitectura de la clase DiVAAlgorithm:

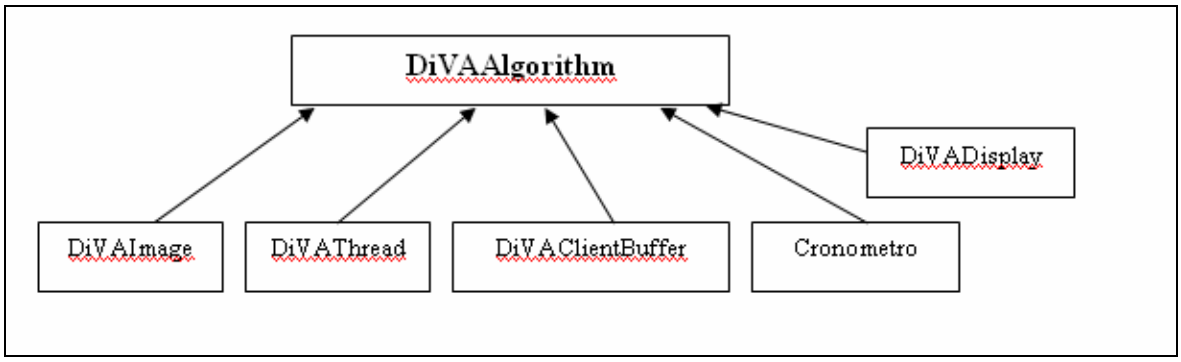


Figura 4-15: arquitectura de la clase DiVAAlgorithm

Como podemos observar, cada clase proporciona una funcionalidad adicional a nuestro componente de procesado. Éstas son:

- DiVAImage. Proporciona las funciones básicas para el manejo de imágenes dentro de la plataforma.
- DiVAThread. Proporciona el funcionamiento cíclico en un hilo independiente.
- DiVAClientBuffer. Nos permite capturar las imágenes de dos modos: secuencial y en modo buffer. Para el modo buffer existe otro hilo que realiza este procesado y comienza a capturar continuamente.
- Cronómetro. En modo depuración/*release*, esta clase nos sirve para tomar medidas de tiempos sobre los distintos procesos que se ejecutan en DiVAAlgorithm.
- DiVADisplay. Esta clase nos permite visualizar resultados intermedios del procesado que se esta llevando actualmente a cabo.

Para proceder a la integración de un nuevo procesado en la plataforma se ha de utilizar el encapsulador de proceso comentado en el Anexo B.6.1 (la clase DiVAAlgorithm). A continuación se describen las características del nuevo componente DiVAObjExtractor:

- Constructor
  - Esta clase tiene un constructor con unos parámetros preestablecidos (dirección IP del servidor, puerto de conexión, almacenamiento de resultados,…) pudiéndose añadir más parámetros a dicho constructor (véase la etiqueta <*parámetros*> .

- Métodos/atributos públicos
  - La clase tiene unos métodos públicos base que servirán como interconexión con los distintos subsistemas. En la clase padre (DiVAAlgorithm) se han implementado con un procesado vacío. Si en cambio se necesita otra funcionalidad diferente, han de ser sobrescritos. Adicionalmente se pueden añadir nuevos métodos/atributos públicos.

- Métodos/atributos privados
  - La estructura básica del encapsulador contiene ciertos atributos privados que permiten el funcionamiento del módulo. Estos atributos básicos se corresponden con un puntero a un fichero de log, punteros a imágenes (en

formato DiVAImage) , un puntero a un conversor de imágenes y al extractor de objetos (ObjExtractor) .

- Procesado de las imágenes
  - Para añadir el procesado, hemos sobrescrito el método *processFrame* de la clase padre (DiVAAlgorithm). Dicho método recibe una imagen DiVA y la procesa, guardando los resultados de manera interna en los atributos de la clase. Estos resultados son accesibles por los demás métodos de la clase, pudiendo enviarlos a las distintas bases de datos disponibles y siendo posible el volcado a disco de las imágenes (la original, la de fondo, la máscara de movimiento, etc.).
  - Ejecución automática del encapsulador de procesado
    - El encapsulador de procesado (DIVAAlgorithm), ejecuta distintas tareas de manera automática (algunas tareas se corresponden con los métodos públicos de la figura XX). La activación/desactivación de estas tareas es controlada por los parámetros del constructor del algoritmo.
    - La secuencia de ejecución cíclica es (del método *process* de la clase padre):
      - Petición de cuadro al sistema de captura (FrameServer)
      - Petición de información al servidor de conocimiento (ContextServer)
      - Petición de información al servidor de datos de análisis (DataServer)
      - Ejecución del método processFrame
      - Almacenamiento de datos en el servidor de datos de análisis, conocimiento (si procede) o fichero a disco (si procede).

Para "aligerar" la carga de proceso de los clientes de cuadros, hemos retirado muchas de las funcionalidades originales de este encapsulador, de modo que el método *processFrame* queda de la siguiente manera (ver figura 4-16):

```
int DiVAObjectExtractor::processFrame(DiVAImage* pImage,void* pdata,void* pContentData)
{

    if(_pImage)
    {
        delete _pImage;
        _pImage = NULL;
    }

    _pImage = pImage->clone();  // Para que el selector disponga de la imagen
                                // actual para procesado

    return 0;

}
```

Figura 4-16: DiVAObjectExtractor

Ya vemos que en realidad, DiVAObjectExtractor prácticamente no tiene mucho mayor funcionalidad que su clase padre, DiVAAlgorithm (el método *process* de ésta es la que se encarga de la recepción de los cuadros procedentes de los DiVAServers).

Inicialmente los clientes eran los que realizaban el procesado (mediante la clase BkgExtractor, para la obtención de la máscara de movimiento, y consecuentemente el porcentaje de movimiento de cada imagen con la anterior), sin embargo se observó que el rendimiento del sistema completo aumentaba si limitábamos la tarea de los clientes a simplemente recibir los *frames* y almacenarlos en buffers y separábamos todo el procesado en otro proceso/programa. De este modo el procesado quedó concentrado en el Selector de cuadros, que explicaremos en la siguiente subsección.

4.2.3.3 Selector de cuadros

4.2.3.3.1 Introducción

Esta función ha sido implementada en la clase DiVASourceSelector. Su función es, por un lado el procesado de las imágenes que reciben los clientes (y a las que accede a través de sus buffers asociados) y a continuación la clasificación de éstas según su relevancia (hablaremos más adelante de los criterios de relevancia de las imágenes). En la versión definitiva del sistema, el selector también se encarga de lanzar el transmisor de imágenes final (*frametransmitter*) que se comunicará con la plataforma CAIN, con vistas realizar el streaming del video final.

Esta clase, puesto que no es un algoritmo que necesite periódicamente capturar imágenes de un frameserver, hereda directamente de DiVAThread (para seguir con la arquitectura de DiVA). Los parámetros de entrada son:

- `int switchTime`: cada switchTime segundos el selector realiza nuevamente una comparación de las prioridades de los cuadros de cada cámara. Dicho de otro modo, cada switchTime segundos puede producirse un cambio de cámara en la imagen visualizada final.
- `int fps`: no utilizado en versiones finales. Inicialmente serviría para limitar el ancho de banda en caso de posibles caídas de la red de transmisión, sin embargo, como en el mejor de los casos la tasa de cuadros por segundo obtenida no es muy alta, este parámetro no es necesario. El programa funciona a la máxima velocidad que le permita la CPU que lo ejecuta.
- `bool renderFrames`: Este valor se debe inicializar a true si se desea que las imágenes recibidas de los clientes se rendericen en una ventana de diálogo. Para ello es necesario específicar la ventana (mediante el parametro dialog).
- `int _algorithm`: Con este “entero” se especifíca cual es el algoritmo, o mejor dicho, “criterio” de selección de imágenes relevantes de entre las disponibles de los clientes conectados. Más adelante especificaremos los criterios disponibles.

- `void* dialog`: Puntero a la ventana de diálogo donde se quiere visualizar las grabaciones de las cámaras y el video final procesado. Esta ventana debe de cumplir unos requisitos que especificaremos en la sección de Integración.

Dado que el selector hereda de la clase DiVAThread, hemos de sobrescribir el método *process*, que como ya vimos en 4.2.2.2 se ejecuta de forma continua, una vez sea lanzado el hilo.

La secuencia de funcionamiento del selector de cuadros es la siguiente (método *process*):

1. **Obtención de imágenes:** esto se realiza cada iteración del bucle *process*. Se piden las imágenes a los buffers de los clientes (siempre se pide el frame con ID consecutivo al que se ha recibido).

2. Si se obtienen imágenes nuevas en cada iteración (dicho de otro modo, si los IDs de las imágenes obtenidas no coinciden con los de las imágenes obtenidas anteriormente):

    a. **Conversión de imágenes a formato openCV:** mediante conversores OpenCVConverter.
    b. **Procesado de las imágenes:** mediante los algoritmos GammaBkgExtractor ya inicializados en el constructor de la clase (`pextractor->processFrame(Frame)`).
    c. **Cálculo de los porcentajes de movimiento:** mediante la máscara de movimiento, que puede ser obtenida a través de los punteros a los algoritmos extractores (`pextractor->getMask()`).
    d. **Obtención de la imagen prioritaria:** se realiza una copia interna de la imagen con más alta prioridad.
    e. **Renderizado de las imágenes (si el parámetro renderFrames esta a true)**
    f. **Transmisión de la imagen:** se introduce la imagen prioritaria en el buffer del transmisor, para ser enviada cuando CAIN se lo pida.
    g. **Liberación de los recursos utilizados**

3. Cada switchTime segundos (esto se controla mediante la clase cronómetro de DiVA):

    a. **Elección del frame prioritario (switchAlgortithm)**
    b. **Actualización de la información de la ventana (si renderFrames esta a true):** esto es: tasa de cuadros por segundo, gráficas de movimiento, porcentajes de movimiento, tamaño de las imágenes, tasa de cambio (switchTime).

A continuación mostramos el código del método *process*:

```
int DiVASourceSelector::process(){      //proceso iterativo, llamado por loop

    int i, ret;
    double dif;
    time_t _Time;
    int motionPixels;
    char* fpsRate=NULL;
    int audio_bitRate = 0;
    int channels = 0;
    int sampleRate = 0;

    OpenCVConverter* pconverter1 = (OpenCVConverter*) _pconverter;

        if(imagen1)
        {
            delete imagen1;
            imagen1 = NULL;
        }
        if(imagen2)
        {
            delete imagen2;
            imagen2 = NULL;
        }
        if(imagen3)
        {
            delete imagen3;
            imagen3 = NULL;
        }
        if(imagen)
        {
            delete imagen;
            imagen = NULL;
        }


    dif = crono->stop();
    if ((dif/1000) >= this->_switchTime){       //Cada cierto tiempo _switchTime "evaluamos" la opción de cambiar de cámara

        //getPriorities();
        switchAlgorithm();              // Aplicamos el algoritmo de cambio de cámara



    if (inicio==0){
        inicio=1;
        for(i=0;i<this->numberOfFolders;i++) mean[i]=priorities[i];     // media(t=0) = valor(t=0)
    }


    if (_renderFrames == 1)     // Actualizamos la informacion
    {
        char cadena[200];
        sprintf(cadena, "%.2lf",(((double)this->fpscount)/((double)this->_switchTime)));
        CString *cad= new CString(cadena);
        ((CDiVASurveillanceSystemDlg*)_dialog)->SetDlgItemText(IDC_EDIT6,LPCTSTR(*cad));

        sprintf(cadena, "camara = %d\n\nprioridades = %lf %lf %lf\n", _id+1,priorities[0], priorities[1], priorities[2]);
        CString char cadena[200] CString(cadena);
        ((CDiVASurveillanceSystemDlg*)_dialog)->SetDlgItemText(IDC_EDIT7,LPCTSTR(*cad1));
        delete cad;     delete cad1;

        ((CDiVASurveillanceSystemDlg*)_dialog)->m_progress1.SetPos((int)(priorities[0]*50));
        ((CDiVASurveillanceSystemDlg*)_dialog)->m_progress2.SetPos((int)(priorities[1]*50));
        ((CDiVASurveillanceSystemDlg*)_dialog)->m_progress3.SetPos((int)(priorities[2]*50));
    }

    for (i=0;i<this->numberOfFolders;i++) priorities[i]= 0;
    this->fpscount=0;
    crono->inicio=0;
    crono->fin=0;
    crono->start();

}

//dif = crono1->stop();
if (inicio==1 ){

    p1->getFromBuffer(&imagen1,id1);            // PIDO IMAGENES
    id1=imagen1->getId(); id1++;
    p2->getFromBuffer(&imagen2,id2);
    id2=imagen2->getId(); id2++;
    p3->getFromBuffer(&imagen3,id3);
    id3=imagen2->getId(); id3++;
```

```
    if (idprev==(id1+1)){}
    else{                           // esto se ejecuta si la imagen recibida del buffer es distinta q la anterior

        IplImage* sampleFrame1=pconverter1->getIplImage(imagen1);
        IplImage* sampleFrame2=pconverter1->getIplImage(imagen2);
        IplImage* sampleFrame3=pconverter1->getIplImage(imagen3);

        if (!_pextractor1)
        {
            _pextractor1 = new GammaBkgExtractor((CvArr*)sampleFrame1,_plog1);
        }

        if (!_pextractor2)
        {
            _pextractor2 = new GammaBkgExtractor((CvArr*)sampleFrame2,_plog2);
        }

        if (!_pextractor3)
        {
            _pextractor3 = new GammaBkgExtractor((CvArr*)sampleFrame3,_plog2);
        }

        //motion detection
        _pextractor1->processFrame(sampleFrame1);
        _pextractor2->processFrame(sampleFrame2);
        _pextractor3->processFrame(sampleFrame3);

        IplImage* mask1 = (IplImage*)(_pextractor1->getMask());
        motionPixels = cvCountNonZero(mask1);
        priorities[0] += motionPixels / (((double)mask1->height)*((double)mask1->width));   // PORCENTAJE DE MOVIMIENTO 1

        IplImage* mask2 = (IplImage*)(_pextractor2->getMask());
        motionPixels = cvCountNonZero(mask2);
        priorities[1] += motionPixels / (((double)mask2->height)*((double)mask2->width));   // PORCENTAJE DE MOVIMIENTO 2

        IplImage* mask3 = (IplImage*)(_pextractor3->getMask());
        motionPixels = cvCountNonZero(mask3);
        priorities[2] += motionPixels / (((double)mask3->height)*((double)mask3->width));   // PORCENTAJE DE MOVIMIENTO 2


    for(i=0;i<this->numberOfFolders;i++) mean[i] = (1/(meancount+1))*(meancount*mean[i] + priorities[i]);   // recalculamos la media
    ret = dumpFrame(_id);   // llamar a dumpFrame, copia el frame "prioritario"

    this->fpscount++;
    this->counter++;
    this->meancount++;
    if (this->counter == _maxFrames) this->counter=0;

    if (_renderFrames == 1)     // Pintamos las imagenes
    {
        refreshDisplay(imagen1,&((CDiVASurveillanceSystemDlg*)_dialog)->m_image_box);   /*******************************/
        refreshDisplay(imagen2,&((CDiVASurveillanceSystemDlg*)_dialog)->m_image_box2);
        refreshDisplay(imagen3,&((CDiVASurveillanceSystemDlg*)_dialog)->m_image_box3);
        refreshDisplay(imagen,&((CDiVASurveillanceSystemDlg*)_dialog)->m_image_box4);
    }

    DiVAImage* pimage = imagen->clone();
    if (txInit==false){ // inicializamos el transmisor
        server = new DiVAStreamServer(this);
        ((DiVAStreamServer*)server)->start();
        txInit=true;
    }
    this->imageBuffer->put(imagen);     // INTRODUZCO IMAGEN FINAL EN BUFFER DE TX

    //Libero memoria
    if(pimage)  delete(pimage);
    if(sampleFrame1)cvReleaseImage(&sampleFrame1);
    if(sampleFrame2)cvReleaseImage(&sampleFrame2);
    if(sampleFrame3)cvReleaseImage(&sampleFrame3);
    idprev=id1;
}
}
return 0;
```

Figura 4-17: código del método *process* de DiVASourceSelector

4.2.3.3.2 Criterios de cambio de cámara.

En el capítulo de estado del arte (sección 2.1.1) ya vimos diferentes enfoques sobre los aspectos importantes en videoseguridad (presencia de objetos móviles y estáticos, movimiento, etc.). Sin embargo en el desarrollo del sistema inteligente, del cual es objeto este PFC, nos hemos centrado exclusivamente en el ámbito de la detección y análisis de movimiento en la escena (sobre todo aprovechando las cámaras y las herramientas de las cuales disponíamos, de las que hablaremos más adelante en el capítulo Integración, Pruebas y Resultados). En particular hemos hecho uso del algoritmo comentado en 2.1.3 para la detección de movimiento.

Para ello, mediante la clase GammaBkgExtractor, se obtiene para cada nueva imagen el número de píxeles diferentes de los correspondientes de la imagen anterior (un cálculo válido para determinar el movimiento en la imagen a "groso modo"). Cada vez que se recibe una nueva imagen se acumula este número con el anterior a modo de "contador", el cual se pone a cero cada switchTime segundos (parece lógico que si la decisión la tomamos cada cierto tiempo, sea con los datos correspondientes a las imágenes recibidas en ese intervalo, y no individualmente con los de la imagen que se reciba al final de ese intervalo).De este modo, las posibilidades de cambio de cámara son:

1. **Por máximo movimiento:** cada switchTime segundos se evalúa el parámetro `priorities[i]` para cada cámara, que a su vez es la suma de porcentajes de movimiento de las imágenes de cada cámara cada switchTime segundos.
2. **Por movimiento superior a un umbral:** se decide cierto umbral THRESHOLD, y se elige la cámara que supere más ampliamente el umbral. Este umbral podría elegirse como en el algoritmo de 2.1.3 ([5]).
3. **Por máximo movimiento con penalizaciones:** con vistas a no monopolizar una cierta cámara en ausencia de movimiento en la escena, puede definirse una cierta penalización PENALTY a aplicar a la cámara que se esté visualizando y así dar la oportunidad a las otras a ser mostradas también.
4. **Por máxima distancia con la media:** de forma iterativa, se va calculando la media móvil de movimiento para cada cámara. Si en un instante dado, el porcentaje de movimiento de la imagen de una cámara es muy distinto al de la media, parece lógico que esa cámara sea candidata a ser visualizada. De este modo, se elige la cámara con mayor diferencia (en valor absoluto) entre el porcentaje de movimiento en un instante y la media.

En la siguiente imagen podemos observar el código de la función de cambio de cámara:

```
void DiVASourceSelector::switchAlgorithm(void)
{
    switch (algorithm){
        case 0:     // máximo de movimiento
            _id = getMaximum(priorities, this->numberOfFolders);
            printf("%d\n", _id);
            break;

        case 1:     // movimiento con penalizaciones
        /// obtener el indice del maximo valor del array
        /// si aplico penalizaciones:
            for (int i=0;i < this->numberOfFolders;i++) priorities[i] = priorities[i]*penalty[i];  // aplico penalizaciones
            _id = getMaximum(priorities, this->numberOfFolders);
            for (int i=0;i<this->numberOfFolders;i++) penalty[i] = 1; // reseteo penalizaciones
            penalty[_id] -= PENALTY;        // esta cámara queda "penalizada" para la siguiente decisión
            break;

        case 2:     // maxima distancia con la media de movimiento
            for (int i=0;i < this->numberOfFolders;i++) priorities[i] = fabs(priorities[i]-mean[i]);  // resto la media
            _id = getMaximum(priorities, this->numberOfFolders);    // mayor diferencia en valor absoluto de la tasa de movimie
            break;

        case 3:     // movimiento por encima de un umbral
            for (int i=0;i < this->numberOfFolders;i++) priorities[i] = priorities[i]-THRESHOLD;  // resto el umbral
            _id = getMaximum(priorities, this->numberOfFolders);    // mayor diferencia
            break;
    }
}
```

Figura 4-18: función de cambio de cámara

4.2.3.4 Transmisor de Imágenes

4.2.3.4.1 Introducción

El selector de cuadros también se encarga de gestionar una herramienta para la transmisión de las imágenes relevantes: DiVAStreamServer (decidimos crear un nuevo componente que aunque fuese lanzado por el selector, tuviese un funcionamiento paralelo a este, por motivos de rendimiento). Este componente es también una clase de C++, que hereda de DiVAThread (ver 4.2.3, figura 4-11). En el constructor se encarga de inicializar y declarar la clase FrameTransmiter, que será la encargada de transmitir las imágenes. A continuación explicaremos el funcionamiento secuencial (función *process*) del componente:

1. **Obtención de los cuadros relevantes:** del selector de cuadros. Este posee un buffer similar a los que poseen los Clientes (DiVAImageBuffer).
2. **Conversión del formato de la imagen:** a formato AVFrame (formato que utiliza y "entiende" CAIN).
3. **Envío de la imagen** (función sendAVFrame).

Este componente dispone de las funciones necesarias para el cambio de formatos de las imágenes DiVA y AVFrame.

```
DiVAStreamServer::DiVAStreamServer(void* selector):DiVAThread()
{
    this->_imagen=NULL;
    idprev=0;
    CodecID audioCodec = CODEC_ID_NONE;
    // inicializamos el transmisor

    _selector=selector;
    DiVAImage* pImage = (DiVAImage*)((DiVASourceSelector*)_selector)->imageBuffer->getSampleElement();
    ((DiVASourceSelector*)_selector)->imageBuffer->get(pImage,0,0);
    id=pImage->getId();
    transmisor = new FrameTransmitter();
    transmisor->setUpTransmission(pImage->getWidth(), pImage->getHeight(),CODEC_ID_NONE, 0, 0, 0);
    if (pImage) delete pImage;

}

int DiVAStreamServer::process()      //proceso iterativo, llamado por loop
{
    if (_imagen){
        delete(_imagen);
        _imagen = NULL;
    }

    _imagen = (DiVAImage*)((DiVASourceSelector*)_selector)->imageBuffer->getSampleElement();
    ((DiVASourceSelector*)_selector)->imageBuffer->get(_imagen,id,0);

    if(!_imagen)()
    else{
        OpenCVConverter* pconverter = (OpenCVConverter*) _pconverter;
        IplImage *pimage = pconverter->getIplImage(this->_imagen);
        int h=pimage->height;
        int w=pimage->width;
        AVFrame frame;
        frame.linesize[0]=w;
        frame.linesize[1]=w/2;
        frame.linesize[2]=w/2;

        frame.data[0]=(uint8_t *)malloc(h*frame.linesize[0]+w);
        frame.data[1]=(uint8_t *)malloc((h/2)*frame.linesize[1]+w/2);
        frame.data[2]=(uint8_t *)malloc((h/2)*frame.linesize[1]+w/2);

        this->IplImage2avframe(pimage,&frame);
        int ret= transmisor->sendAVFrame(&frame,0,0,0);

        free(frame.data[0]);
        free(frame.data[1]);
        free(frame.data[2]);

        if(pimage)
            cvReleaseImage(&pimage);

        idprev = id;
    }
    return 0;
}
```

Figura 4-19: métodos constructor y process de DiVAStreamServer

4.2.3.4.2 Clase *FrameTransmitter*

Esta clase de C++ no hereda de ningún componente de DiVA, pero se encarga de ejecutar un servidor de descripciones de imágenes de tipo DiVAServer, que será el que reciba peticiones de la plataforma CAIN. En la inicialización (en el constructor) se encarga de lanzar el transmissionServer en un determinado puerto y con un determinado tamaño de buffer de descripciones y otras características propias de la clase padre (DiVAServer). En el método setUpTransmission se copian los parámetros de inicialización relacionados con audio (los codecs, el bitrate, que por las características y limitaciones de este PFC, no utilizaremos), el tamaño de las imágenes, etc.

```
FrameTransmitter::FrameTransmitter(void) {
    this->bufferElement = new Description(AVF_DESCRIPTION_SIZE);
    this->descriptionBuffer = new DescriptionBuffer(AVF_DESCRIPTION_SIZE,AVF_BUFFERSIZE,AVF_SERVER_TIMEOUT,AVF_SERVER_CONSUMECONTROL,AVF_SERVER_M
    this->transmissionServer = new DiVAServer(NULL, this->descriptionBuffer, AVF_SERVER_PORT);
    this->transmissionCounter=1;
    this->transmissionServer->start();

    this->transmissionSetupFlag = false;
    this->AVFwidth=0;
    this->AVFheight=0;
    this->audio_bitRate=0;
    this->channels=0;
    this->sampleRate=0;
}

void FrameTransmitter::setUpTransmission(int AVFwidth, int AVFheight, CodecID audioCodec, int audio_bitRate, int channels, int sampleRate) {

    printf("\n[FrameTransmitter] Transmission Parameters set:");
    this->AVFwidth=AVFwidth;
    this->AVFheight=AVFheight;
    this->audioCodec=audioCodec;
    this->audio_bitRate=audio_bitRate;
    this->channels=channels;
    this->sampleRate=sampleRate;
    this->transmissionSetupFlag = true;
    printf("\n\tFrame: %dx%d, \ntAudio[Codec: %d, bitrate:%d, channels:%d, sampleRate:%d]",
        this->AVFwidth, this->AVFheight, this->audioCodec, this->audio_bitRate, this->channels, this->sampleRate);
}
```

Figura 4-20: inicialización del FrameTransmitter

Esta clase posee la función SendAVFrame, que es llamada por el DiVAStreamServer, que se encarga de insertar la imagen del Selector de cuadros en el buffer del Servidor de descripciones, que será enviada cuando un cliente remoto de tipo DiVAClient lo demande. Aquí se introduce la imagen en el campo videoBuffer, se rellenan la información de tamaño de imágenes, información de audio, etc; se serializa todo ello y se introduce en el buffer de descripciones (`descriptionBuffer->put(bufferElement))` . La figura 4-21 muestra el código de esta función.

```
bool FrameTransmitter::sendAVFrame(AVFrame *frame, int16_t *audioBuffer, int audioSampleLength, int audioSamples) {

    if(this->transmissionSetupFlag==false) {
        printf("\n[FrameTransmitter::sendAVFrame] Error: transmission parameters must be set before transmission");
        return false;
    }

    AVFDescriptor frameDescriptor;
    //reservamos memoria
    frameDescriptor.videoBuffer[0]=(uint8_t*)malloc(frame->linesize[0]*this->AVFheight*sizeof(uint8_t));
    frameDescriptor.videoBuffer[1]=(uint8_t*)malloc(frame->linesize[1]*this->AVFheight/2*sizeof(uint8_t));
    frameDescriptor.videoBuffer[2]=(uint8_t*)malloc(frame->linesize[1]*this->AVFheight/2*sizeof(uint8_t));
    frameDescriptor.audioBuffer=(int16_t*)malloc(audioSamples*this->channels*sizeof(int16_t*));

    //metemos la informacion de tamaño de frame,codec+parametros de audio,etc
    frameDescriptor.width = this->AVFwidth;
    frameDescriptor.height = this->AVFheight;
    frameDescriptor.audio_bitRate = this->audio_bitRate;
    frameDescriptor.audioCodec=this->audioCodec;
    frameDescriptor.sampleRate=this->sampleRate;
    frameDescriptor.channels=this->channels;
    frameDescriptor.audioSamples=audioSamples;
    frameDescriptor.audioLength=audioSampleLength;
    frameDescriptor.linesize[0]=frame->linesize[0];
    frameDescriptor.linesize[1]=frame->linesize[1];
    frameDescriptor.linesize[2]=frame->linesize[2];
    frameDescriptor.linesize[3]=frame->linesize[3];

    memcpy(frameDescriptor.videoBuffer[0], frame->data[0], frame->linesize[0]*this->AVFheight);
    memcpy(frameDescriptor.videoBuffer[1], frame->data[1], frame->linesize[1]*this->AVFheight/2);
    memcpy(frameDescriptor.videoBuffer[2], frame->data[2], frame->linesize[2]*this->AVFheight/2);
    memcpy(frameDescriptor.audioBuffer, audioBuffer, audioSamples*this->channels*2);

    //envio de la informacion
    this->bufferElement->setId(this->transmissionCounter++);
    int tamanoPaquete=0;
    char *bufferEmpaquetado = serializeAVFDescriptor(frameDescriptor, tamanoPaquete);
    if(tamanoPaquete==0 || bufferEmpaquetado==NULL) {
        printf("\n[FrameTransmitter::sendAVFrame] Error: Serialization failed");
        return false;
    }

    this->bufferElement->setBuffer(bufferEmpaquetado, tamanoPaquete);

    bool retorno=false;
    for(int sendIteration=0; sendIteration<AVF_SERVER_MAX_NUMBER_ITERATIONS; sendIteration++) {
        int ret = this->descriptionBuffer->put(this->bufferElement);
        if(ret == 0)  { //envio correcto
            //printf("\n[FrameTransmitter::sendAVFrame] AVF Transmission correct[%d]", this->transmissionCounter);
            retorno = true;
            break;
        } else {
            printf("\n[FrameTransmitter::sendAVFrame] Transmission try %d failed", sendIteration);
        }
    }

    free(frameDescriptor.videoBuffer[0]);
    free(frameDescriptor.videoBuffer[1]);
    free(frameDescriptor.videoBuffer[2]);
    free(frameDescriptor.audioBuffer);
    free(bufferEmpaquetado);

    return retorno;
}
```

Figura 4-21: función SendAVFrame

4.3 Modificaciones / Implementaciones basadas en CAIN para la creación del sistema de streaming

4.3.1 Introducción

El motor CAIN ha sido diseñado para realizar adaptación a todos los niveles posibles (a nivel de contenidos, de sistemas, de estructura, de diseño, etc.). Dado que CAIN es un motor de adaptación basado en metadatos, hace uso de anotaciones de los recursos para mejorar las anotaciones. Específicamente, el contenido multimedia es anotado según el estándar MPEG-7 [14], y el sistema de adaptación hace uso de la descripción del *framework* MPEG-21 [15].

Como vimos en 3.3.2 y según lo definido en [15], las entradas y salidas de CAIN están representadas a través de sus DIs (Digital Ítems), ver figura 4-22 . La razón de la elección de los DIs es que son capaces de representar virtualmente cualquier tipo de contenido o composición. En adelante utilizaremos el término **resource** (recurso) para referirnos al fichero o medio referenciado por el elemento *Resource.*

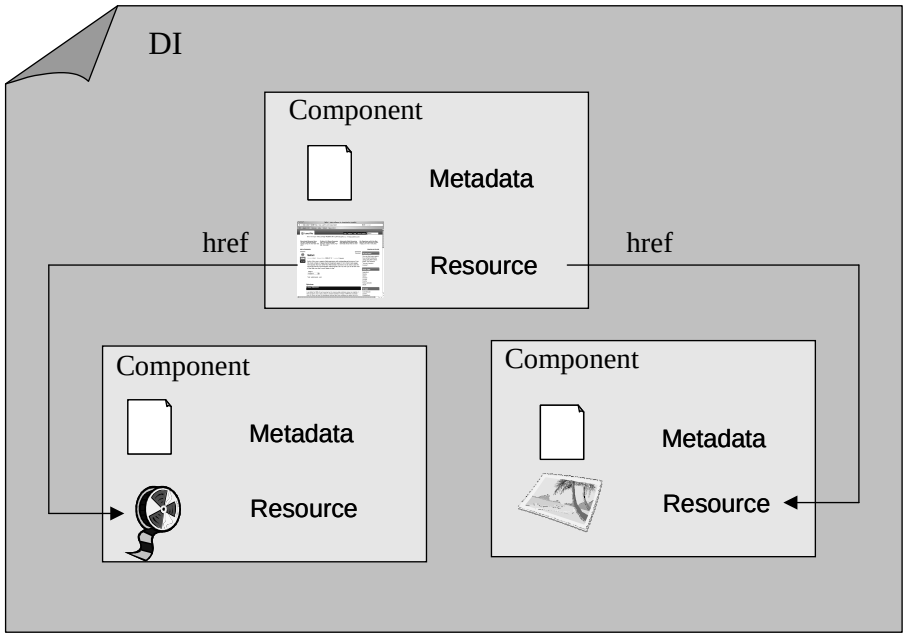


Figura 4-22: composición de los DIs en CAIN

Utilizaremos el término “adaptación a nivel DI” para referirnos a la adaptación realizada por CAIN sobre un DI, en donde la entrada de la adaptación es un DI y la salida del proceso de adaptación se representa como DI adaptado (*adapted DI*). Dado que un DI puede contener más de un componente (*Component*), la adaptación será “a nivel de componente” mediante los módulos software llamados CATs (*Content Adaptation Tools*) como vimos anteriormente en 3.3.1. También se han considerado casos como el de la figura 4-22, donde un *Component* (la página HTML) puede contener referencias a recursos de otros *Components.*

Los metadatos pueden ser extraídos automáticamente por los CATs, o pueden ser introducidos como *Components* de entrada. La adaptación a nivel de *Component* consta de dos fases: en la primera el recurso se obtiene del repositorio de contenidos. Si además se realiza adaptación semántica, el recurso y los metadatos asociados son analizados. En la segunda etapa los metadatos se introducen en el *Descriptor* del *Component* y la información resultante de los análisis de los contenidos es usada para la adaptación (transcodificación, transformación, o *transmode*).

La plataforma de CAIN está dividida, como ya hemos visto, en módulos (*modules*). En la figura 4-23 se puede apreciar esta división y la relación entre ellos. Estos *modules* son partes arbitrarias de un determinado componente (término utilizado para designar un sistema de software autónomo). Mientras un componente puede realizar en un proceso aparte, el módulo se ejecuta dentro del componente "contenedor". Este diseño permite la comunicación entre módulos.

La librería que contiene las funciones para implementar la adaptación (transcodificación, transformación, de imágenes y video, extracción de imágenes, truncado de video, etc.) es la **Tlib**. Los módulos que realizan la "adaptación a nivel de componente", son los CATs. Los CATs se basan en la librería Tlib para realizar partes de la adaptación de los recursos descritos en los *Components* (y la adaptación de los metadatos asociados, aunque para ello no se basen en la Tlib).

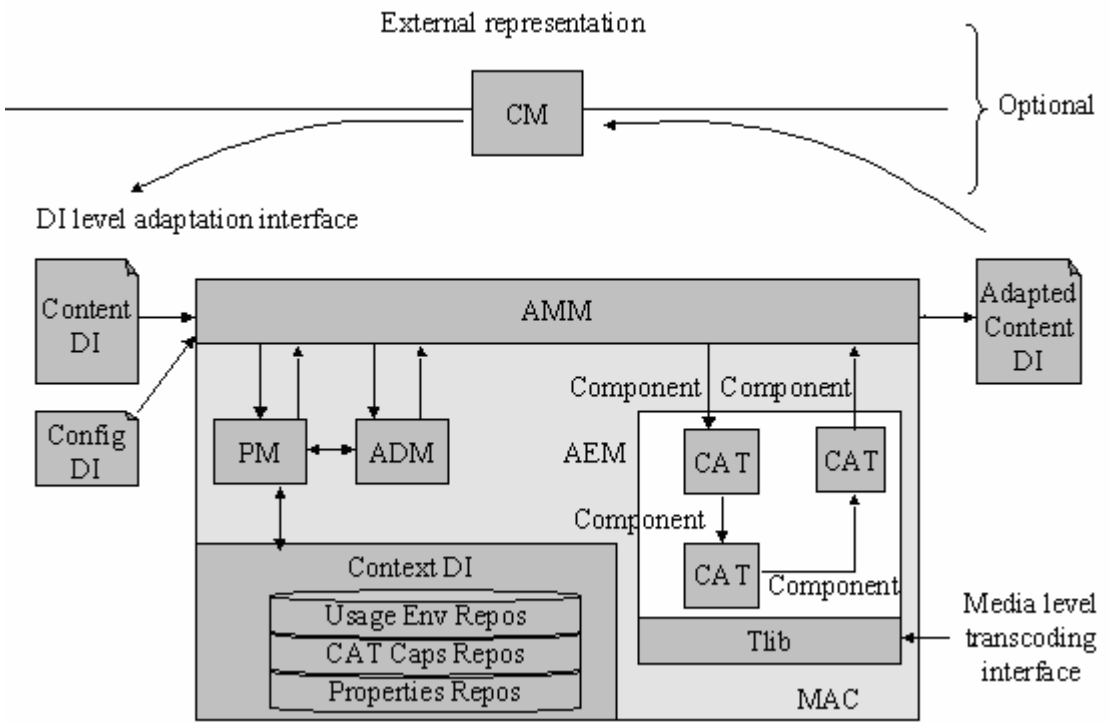


Figura 4-23: Diagrama de módulos de CAIN

El módulo de análisis sintáctico **PM (***Parsing Module***)** se encarga de cargar y leer la información del **Content DI (**DI de contenidos), **Configuration DI (**DI de configuración) and **Context DI** (de contexto). Dentro del PM podemos encontrar el **CM**

(*Coupling Module*), encargado de ofrecer interfaces a otros componentes mediante tecnología externa (no MPEG-21) para representación de contenidos multimedia (p.e. HTML, SMIL o NewsML) y transformarlos en DIs de entrada de acuerdo con el estándar MPEG-21. Además es capaz de transformar el DI de salida adaptado en su forma de representación externa.

Cuando llega el DI, el **ADM** (*Adaptation Decision Module*) se encarga de decidir la secuencia de conversiones necesaria para transformar el DI de entrada en el DI adaptado, teniendo en cuenta los DIs de configuración y contexto. El módulo encargado de coordinar la ejecución de los CATs (que ejecutan secuencialmente) es el **AEM** (*Adaptation Execution Module*). El **AMM** (*Adaptation Management Module***)** se encarga de coordinar el proceso completo de adaptación a nivel DI. El resto de módulos situados debajo del módulo AMM en la figura 4-23 realizan las tareas que pide el AMM. Antes de que se realice la adaptación, los CATs involucrados en ella deben de estar instalados en el AEM y su ficheros de *Capabilities* (capacidades de conversión) deben estar registrados en el *CAT Capabilities Repository* (repositorio de capacidades de los CAT). El resto de información acerca de las características de la red, terminales y otras preferencias deben registrarse en el *Usage Environment Repository* (repositorio de uso en el entorno).

El módulo AEM como hemos dicho se encarga de llevar a cabo la adaptación (en algunos casos, adaptaciones en múltiples fases). Recordamos que la adaptación en CAIN se realiza a nivel de componente. Un DI se compone de varios componentes, donde cada uno de estos contiene un único recurso y varios descriptores (estos últimos son opcionales y contienen información del recurso, de sus variaciones y de su lenguaje). Cuando un CAT se ejecuta para realizar adaptación, este opera sobre uno o varios *Components* del DI. El CAT realiza las siguientes operaciones sobre cada componente:

1. Lee el recurso y los eventuales descriptores.
2. Adapta el recurso de acuerdo a los parámetros proporcionados.
3. Modifica o crea los descriptores que describen el recurso adaptado.
4. Crea el componente adaptado con el recurso adaptado y sus descriptores adaptados.

La comunicación entre los CATs se detalla en el fichero de capacidades de cada CAT, y puede tener los siguientes valores, que presenta la tabla 4-1.

Mecanismos de vinculación	Descripción n
urn:mpeg:mpeg21:2007:01-BBL-NS:handler:INPROCESS	Técnica utilizada para transferir información entre CATs durante el proceso. En este caso, los objetos de CAIN se transfieren en memoria según el modelo *Pull.*
urn:mpeg:mpeg21:2007:01-BBL-NS:handler:FILE	Se puede leer/escribir los ficheros proporcionados mediante una URL
urn:mpeg:mpeg21:2007:01-BBL-NS:handler:TCP	Se puede leer/escribir los sockets TCP proporcionados mediante una URL
urn:mpeg:mpeg21:2007:01-BBL-NS:handler:HTTP	Se puede leer/escribir por el protocolo HTTP. La IP y el puerto se proporcionan en la URL
urn:mpeg:mpeg21:2007:01-BBL-NS:handler:RTSP	Se puede leer/escribir por el protocolo RTSP. La IP y el puerto se proporcionan en la URL

Tabla 4-1: Mecanismos de vinculación entre CATs soportados.

Esta tabla ilustra los distintos casos de comunicación entre CATs. El primer CAT de una secuencia de conversión siempre lee el recurso (por ejemplo. `file:///videos/newsreport2.mpg`) y la última conversión lo guarda en disco. En los casos donde existe la conversión en una secuencia y el modo OnA está activo, la primera conversión crea un stream INPROCESS, que se genera progresivamente del recurso. De esta manera, solo una primera parte del archivo debe de ser leída antes de que el resto de la secuencia de conversión pueda empezar su tarea. Del mismo modo, el CAT final de la secuencia de conversión se encarga de escribir el recurso a disco o de forma alternativa por la red por medio de un protocolo de streaming (por ejemplo HTTP o RTSP).

4.3.2 Desarrollo del Sistema de Streaming.

En la figura 4-24 puede observarse el diagrama con los módulos y DIs involucrados en la ejecución del Sistema de Streaming, a través del motor de CAIN. En la subsección introductoria anterior hemos explicado el funcionamiento y finalidad de cada uno de los módulos y componentes que aparecen en la figura.

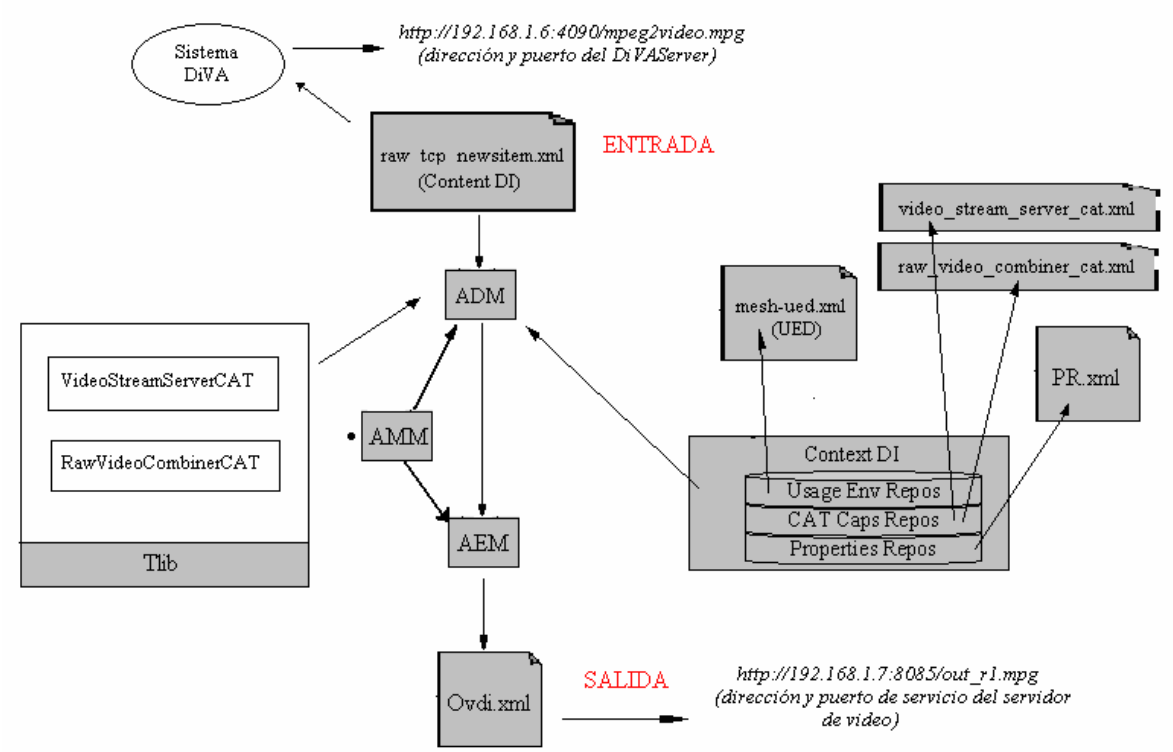


Figura 4-24: diagrama del servidor de video de CAIN

Dado que tenemos que recibir las imágenes procesadas por el Sistema Inteligente de Videoseguridad, basado en DiVA, introduciremos como entrada la dirección y puerto de servicio del DiVAServer integrado en el Sistema. El CAT RawVideoCombiner tiene integrado un DiVAClient, que es el que se comunica con el DiVAServer del Sistema Inteligente. Esta dirección y puerto se detallan en el **Content DI** (llamado raw_tcp_newsitem.xml). Por otra parte, el **Context DI** se forma a través de los repositorios de Propiedades, Capacidades y uso en el entorno (**Properties, CAT Caps & Usage Env)**. En dichos repositorios se registran la información contenida en el mesh-ued.xml (UED, indica los posibles terminales y sus características), los ficheros de los CATs involucrados (que registran sus *capabilities* en el repositorio correspondiente) y el fichero PR.xml (fichero de propiedades). Todos estos DI forman el Context DI.

El ADM se basa en el Context DI y el Content DI para tomar la decisión de la adaptación a realizar. La decisión pasa al AEM, quien se encarga de realizar la secuencia de conversiones, que tiene lugar según el siguiente diagrama:

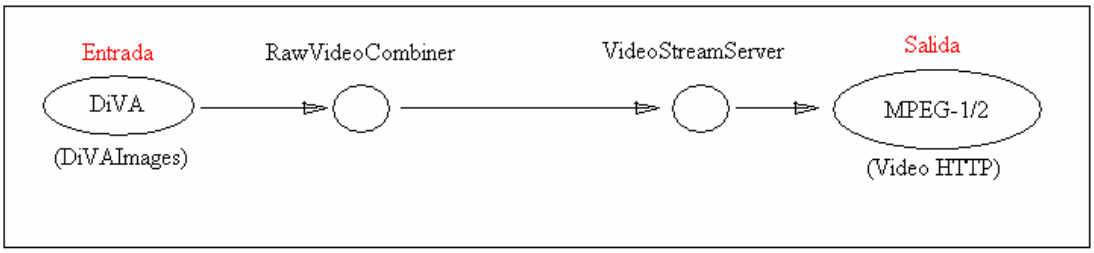


Figura 4-25: secuencia de conversiones

El AEM entonces se encarga de la llamada a los CAT, según el orden establecido en la figura 4-25. El recurso de salida, por tanto, es video HTTP codificado en MPEG-1/2 (u otros) según especifiquemos en el UED (fichero mesh-ued.xml en nuestro diseño). El fichero Ovdi.xml (*Output Video Digital Item*) representa el DI adaptado, con las características del recurso adaptado (el *Component* adaptado). Dentro de este fichero podemos encontrar la referencia al stream, en el campo *Resource mimeType* , donde se encuentra la dirección IP y puerto donde se sirve el stream de video HTTP.

4.3.2.1 RawVideoCombiner CAT

Como muestra la figura 4-26 el RawVideoCombinerCAT es de tipo `PullableMemoryCAT` (CAT que recoge los datos de memoria, en este caso de un socket, en lugar de acceder a disco, lo que aumenta la velocidad de acceso a los datos) que lee video sumarizado sin comprimir y produce video MPEG-2. El CAT acepta el mecanismo de vinculación de entrada :TCP (DIVA frame server, precisamente el utilizado en nuestro proyecto) y la vinculación de salida es :INPROCESS. De este modo, el escenario de ejecución del CAT es el siguiente:

1. El CAT recibe video sin comprimir del DiVAServer por medio del mecanismo :TCP.
2. El CAT produce un archivo MPEG-2 por medio del mecanismo :INPROCESS.

La clase *RawInputStream* es del tipo *InputStream* utilizado para producir la salida :INPROCESS del RawVideoCombinerCAT.

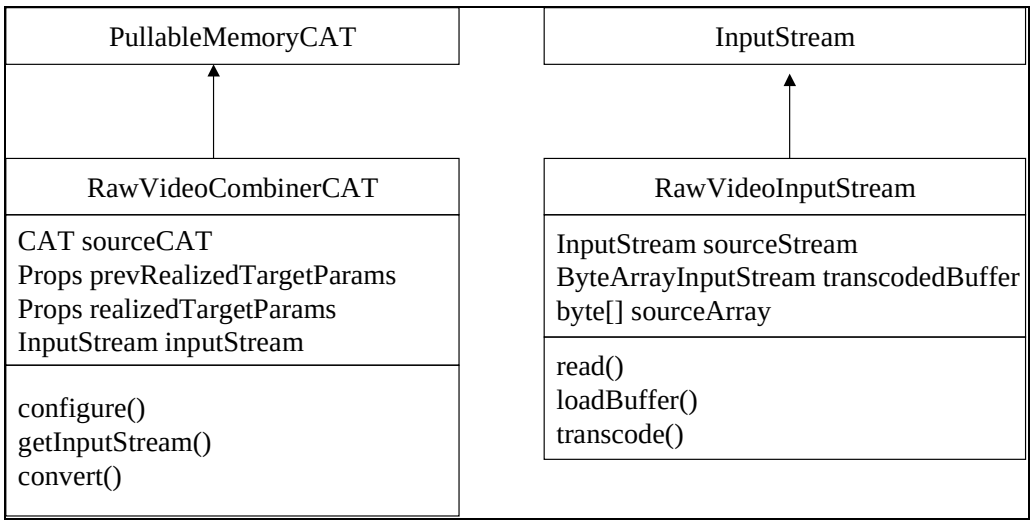


Figura 4-26: Diagrama de las clases que conforman el RawVideoCombinerCAT

El método transcode() implementado en RawVideoInputStream accede a las librerías de *ffmpeg* [17] para realizar la transcodificación del recurso.

Para la prueba de estas clases utilizamos el programa RawVideoCombinerTestCase, el cual veremos más adelante en la sección de integración (5.2.3).

5 Integración, pruebas y resultados

5.1 Introducción

En este capítulo hablaremos de cómo se han integrado las distintas tecnologías que componían nuestro diseño del sistema de videoseguridad y el resultado de dicha unión. En primer lugar cabe recordar dicho diseño, que puede mostrarse en la siguiente figura:

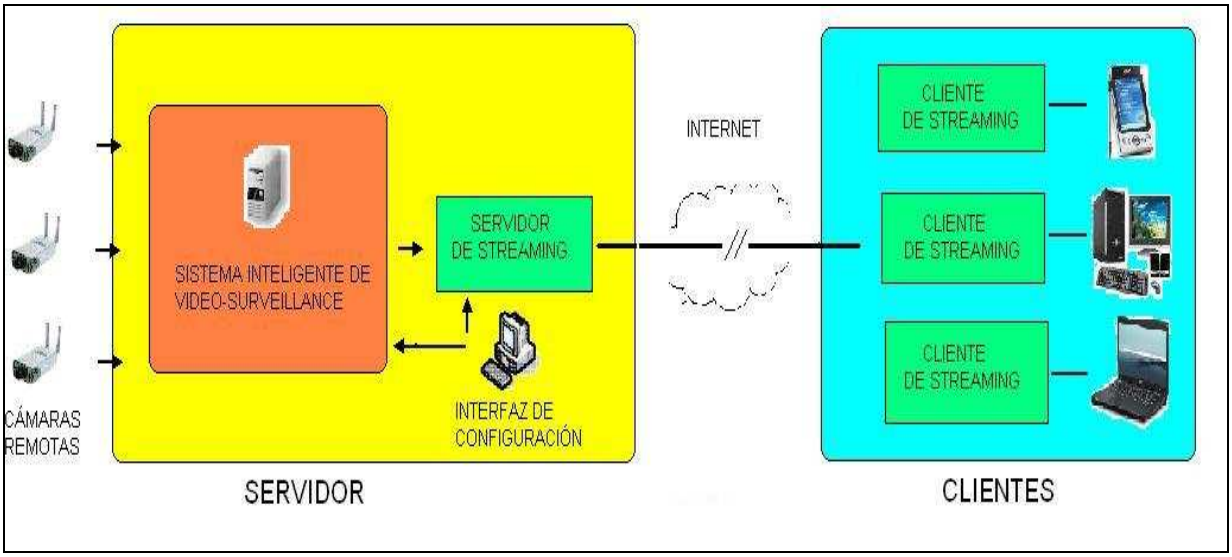


Figura 5-1: Diseño generalizado del sistema de videoseguridad

Como ya vimos en el capítulo anterior el sistema inteligente de video-surveillance (en color naranja) ha sido implementado por medio de la plataforma DiVA, mientras que el servidor de streaming (a la izquierda, de color verde) tiene una pequeña parte de DiVA y la otra gran parte su funcionamiento lo lleva a cabo el motor de adaptación CAIN.

A continuación veremos más a fondo como están integradas estas dos plataformas, y finalmente como queda distribuido el sistema de la figura 5-1.

5.2 Integración

En la figura 5-2, que no es más que la 5-1 ampliada con la información de CAIN, podemos apreciar cual es la división en módulos software de todo este proyecto:

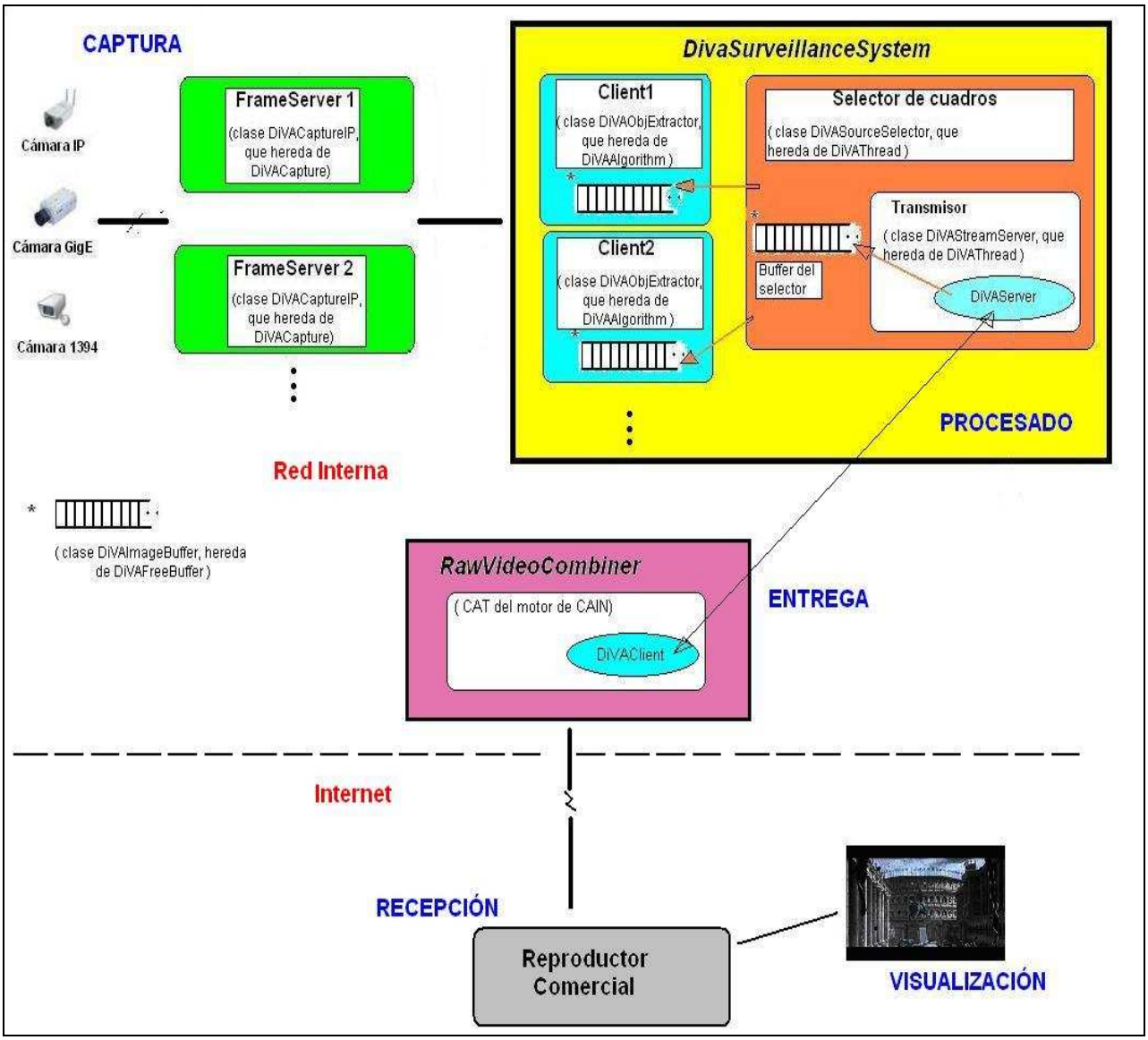


Figura 5-2: Esquema del sistema de vigilancia al completo

En esta figura puede apreciarse que la plataforma de procesado y la de entrega se comunican mediante los módulos de DiVA de entrega de imágenes, como son DiVAServer y DiVAClient. A continuación, en las siguientes subsecciones explicaremos como se encapsulan estos módulos software en programas independientes.

5.2.1 *Frameservers*

Como ya vimos en la sección 4.2.3.1, el módulo software *frameserver* se encargaba de la captura, almacenamiento (en buffers ad hoc) y distribución de imágenes captadas por distintas fuentes, que podían ser cámaras IP, 1394 (Firewire), webcam USB ó de archivos de vídeo (en nuestro proyecto utilizamos como prueba el *frameserverIP*, por razones que detallaremos en la sección de Pruebas y resultados 5.3).

Para poder utilizar estos *frameservers*, se ha compilado la solución de Visual Studio DiVAFRAMESERVERS, que "produce" los ejecutables de los servidores de imágenes. A

continuación por cada cámara que queramos involucrar en el sistema hemos ejecutado un frameserver, con las características de cada cámara en su fichero de configuración correspondiente (ver Anexo A.1.1.1).

Estos frameservers pueden lanzarse en la misma máquina que ejecutará el sistema de procesado, o en una independiente, para mejorar el rendimiento conjunto.

En la siguiente figura 5-3 puede observarse la apariencia de la interfaz de los *frameservers:*

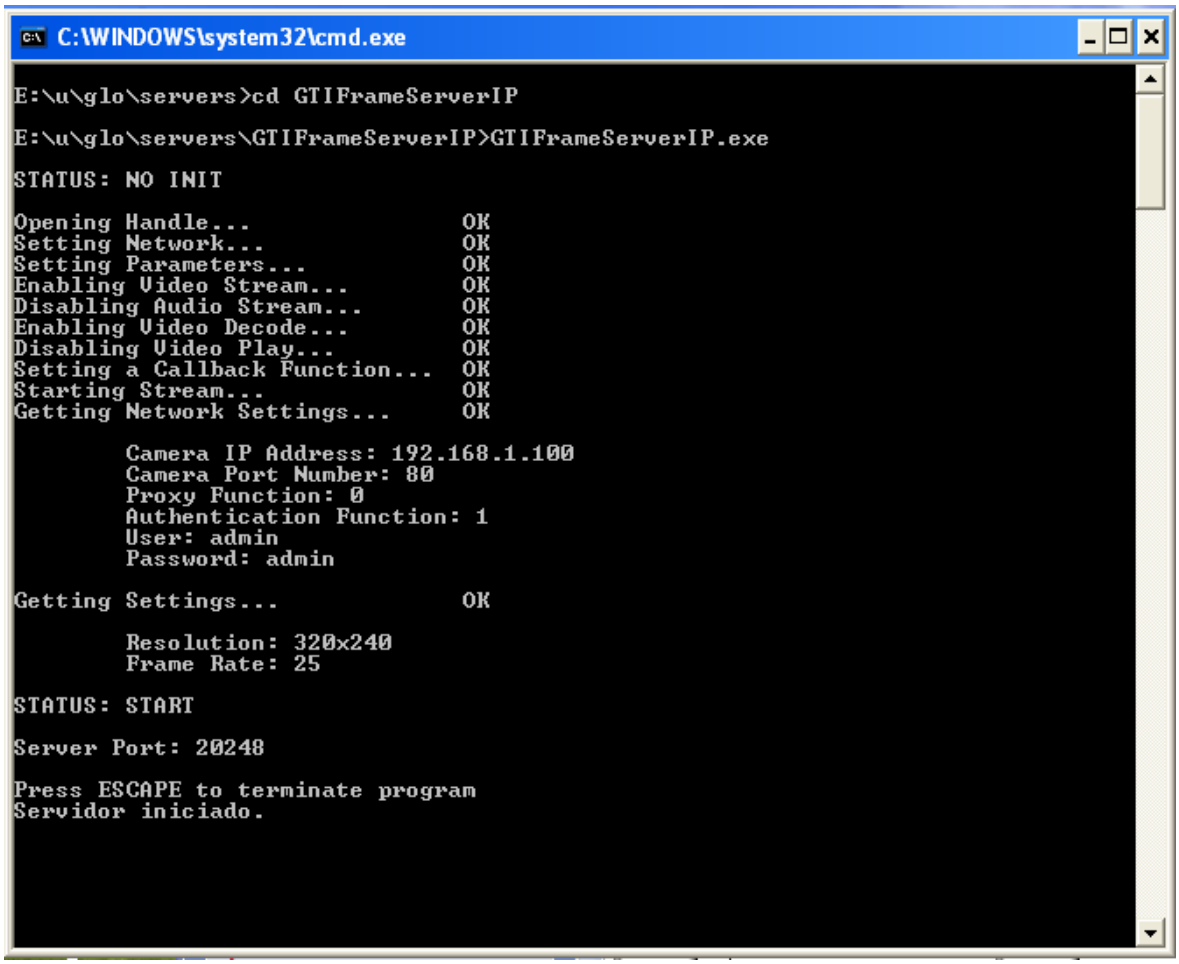


Figura 5-3: Apariencia de los *frameservers*.

Aquí se nos informa de las características del servidor: puerto de servicio, tasa de cuadros máxima, resolución de las imágenes captadas, etc.

5.2.2 DiVAMultiSurveillanceSystem

Como ya vimos en la subsección 4.2.3, el sistema inteligente se componía de tres módulos que ejecutaban de forma paralela (*threads*), a saber: los clientes de imágenes, el selector (procesado) y el transmisor. Para poder obtener el ejecutable del sistema inteligente, hemos compilado el proyecto DiVAMultiSurveillanceSystem (que integra a los anteriores), que produce la aplicación de mismo nombre. Esta aplicación a la que a partir de ahora llamaremos Sistema de Procesado, permite al usuario a través de su interfaz por un lado la configuración de ciertos aspectos del funcionamiento del sistema y la visualización de la grabación de las cámaras y del video de salida.

En la figura 5-4 podemos ver una captura de pantalla de la interfaz de la aplicación (para 3 cámaras IP):

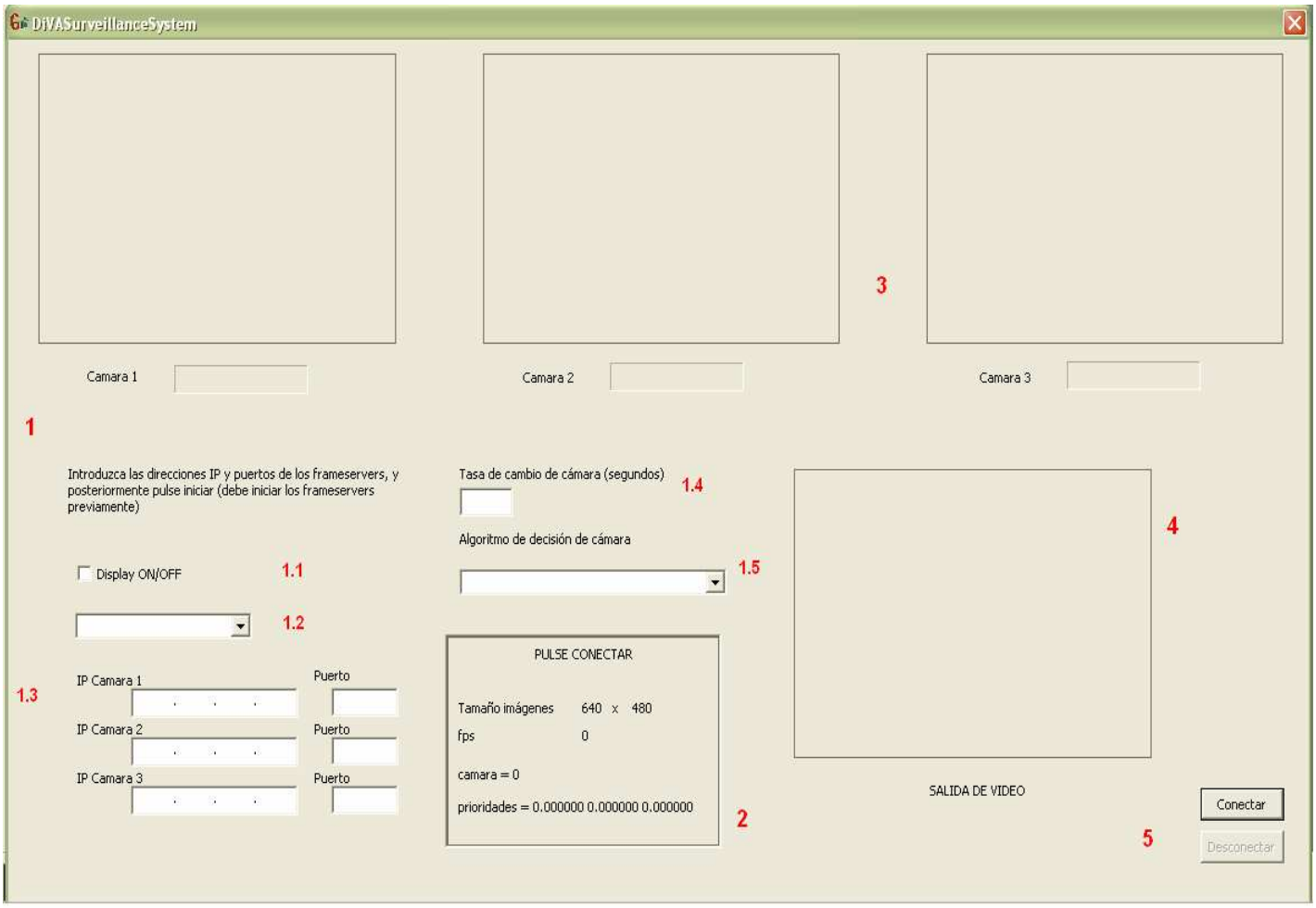


Figura 5-4: Interfaz de la aplicación DiVAMultiSurveillance system.

1. **Datos de configuración**:

Pestaña de display: permite apagar o activar las ventanas de vídeo de la interfaz. Desactivar esta opción aumenta el rendimiento del selector, lo que permite aumentar la tasa de cuadros de salida.

Configuración: esta pestaña ofrece la opción de introducir automáticamente la configuración por defecto (ahorra tiempo a los programadores, rellena por defecto la configuración para pruebas en el laboratorio del VPU en la EPS, con los *frameservers* funcionando en la misma máquina).

Configuración de las cámaras: si la configuración no es por defecto, el sistema leerá las direcciones IP y puertos de los frameservers (asociados a cada cámara) introducidas por el supervisor.

Tasa de cambio: en segundos, es el tiempo máximo que una cámara puede visualizarse, en el caso de que haya sido previamente seleccionada, y sus imágenes ya no sean "relevantes". En la configuración por defecto, esta tasa es de 2 segundos.

Algoritmo de decisión: permite cambiar el algoritmo de cambio de cámara, o criterio de selección de imágenes relevantes. Para más información acerca de este parámetro, consultar la sección 4.2.3.3.2. Por defecto el algoritmo seleccionado es el de máximo movimiento.

2. **Información al supervisor**

   En el pequeño recuadro mostrado en 2 puede observarse la información mostrada al usuario supervisor, que incluye:

   - Tamaño de las imágenes mostradas
   - Tasa de cuadros por segundo mostrada (como vimos en 4.2.3.3.1, esta tasa es la máxima que admite el sistema, a no ser que el programador la limite en el código mediante el campo fps del selector).
   - Cámara seleccionada en cada instante por el selector (puede verse su imagen en la ventana contigua, 4).
   - Prioridades de cada cámara, cada "Tasa de Cambio" segundos, en forma numérica con 6 dígitos.

3. **Imágenes de los frameservers**

   En estas ventanas podemos visualizar (si la pestaña de Display On/Off está activa) la grabación de las cámaras que supervisan los *frameservers*. Para

que el supervisor pueda realizar una comparación más sencilla de la diferencia de movimiento de las cámaras, se han incrustado por cada cámara una barra de progreso que nos permite valorar la prioridad de las imágenes que se visualizan en cada instante.

4. **Salida de video**

   En esta ventana se puede observar el video que se transmite por la red (el que teóricamente, con un retardo que depende de la red de transmisión, se debería estar visualizando en el reproductor remoto), que es el resultado de todo el procesado realizado por el selector.

5. **Botones de conexión**

   Una vez pulsado el botón de conexión, se lee la información de configuración y se inicia el sistema, pudiendo ser interrumpida la ejecución de éste para cambios en la configuración (NOTA: esto interrumpiría la ejecución del cliente incrustado en CAIN, lo que haría que se detuviese la transmisión del video).

Hasta aquí la parte de la interfaz con el usuario. En el Anexo A se explicará el software que es necesario instalar para poder compilar y ejecutar esta aplicación.

5.2.3 Servidor de video HTTP (CAIN)

Según hemos visto en la parte de desarrollo, la misión del motor de CAIN en este Proyecto Fin de Carrera es servir el video procesado que proviene del Sistema de Procesado de forma que sea accesible a través de la red. Realiza, por tanto, una adaptación de las imágenes con formato de DiVA (rigurosamente hablando, con formato de OpenCV), transformándolas en video HTTP, como indica la figura 5-5:

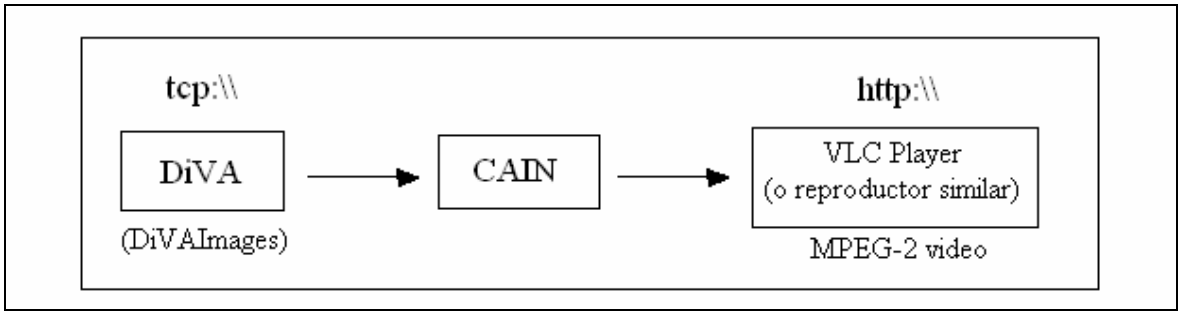


Figura 5-5: adaptación realizada

Esta labor se lleva a cabo ejecutando un programa en Java que se encarga de llamar al motor de CAIN con todas las propiedades y DIs que aparecen en la figura 4-24 en la

sección de Desarrollo (4.3.3). Este programa es el *RawVideoCombinerTestCase.java*, que podemos ver en la figura 5-6:

```
public class RawVideoCombinerTestCase {

    /**
     * Read a raw video from DIVA frame server and produces an HTTP MPEG-2 video
     */
    public static void main (String[] args) throws Exception {
        // 1. CREATE THE AMM
        String ued_url = Config.getFileProtocol(Config.getWorkingDir()+File.separator+"mesh-ued.xml");
        List<String> cat_capabilities_urls = new ArrayList<String>();
        cat_capabilities_urls.add(Config.getFileProtocol(Config.getWorkingDir()+File.separator+"video_stream_server_cat.xml"));
        cat_capabilities_urls.add(Config.getFileProtocol(Config.getWorkingDir()+File.separator+"raw_video_combiner_cat.xml"));

        AMM amm = new AMM(ued_url,cat_capabilities_urls,Config.getFileProtocol(Config.getWorkingDir()+File.separator+"pr.xml"));
        PropsRepos propsRepos = amm.getPM().getPropsRepos();

        // 2. EXECUTE THE IVDI ADAPTATION
        String input_di_url = Config.getFileProtocol(Config.getWorkingDir()+File.separator+"raw_tcp_newsitem_di.xml");
        String output_di_url = Config.getFileProtocol(Config.getWorkingDir()+File.separator+"ovdi.xml");
        ARC arc = new ARC("mpeg2_adapted_online_web");// Terminal destino
        // Executes the adaptation
        amm.adapt(input_di_url,"summarization",output_di_url,arc);

        // Sleep the thread to impide the finalization of the test
        Thread.sleep(1000000000);
    }
}
```

Figura 5-6: Código del programa RawVideoCombinerTestCase.java

Podemos ver que la ejecución secuencial del programa es idéntica a la descrita en 4.3.3. Inicialmente se cargan (se leen) las propiedades de los DIs de propiedades: el *mesh-ued*.xml (propiedades de terminales), los ficheros de capacidades de los CATs que intervienen (*RawVideoStreamServer* y *VideoStreamServer*) y el fichero de propiedades, PR.xml. A continuación se crea el **AMM** (Adaptation Management Module) con las propiedades anteriormente descritas. Después se especifican los **Content DI** y **Adapted Content DI** (raw_tcp_newsitem_di.xml y ovdi.xml), y se especifica el terminal destino al **ARC** (*Adaptation Request Configuration*). Finalmente, con todos estos datos se ejecuta el módulo **Adapt(..)** del AMM, que será el encargado de invocar al **AEM** y este a su vez a los CATs correspondientes para realizar la secuencia de conversiones.

El video producido HTTP puede ser reproducido mediante un reproductor de video que soporte streaming HTTP en formato MPEG-1/2, simplemente introduciendo como dirección y puerto del recurso fuente el definido en el DI de salida (ovdi.xml, ver Anexo D.3).

5.3 Pruebas y resultados

En esta sección hablaremos de las pruebas realizadas en el VPU Lab, de la Escuela Politécnica Superior de la UAM, con los medios que éste ofrece. Por ello en primer lugar hablaremos de las cámaras utilizadas.

5.3.1 Cámaras utilizadas

Como ya dijimos en anteriores secciones, el programa *DiVAFrameServerIP* es un *FrameServer* que utiliza las cámaras PTZ instaladas en el hall de la Escuela Politécnica Superior para capturar imágenes y señal de vídeo.

Las cámaras utilizadas son el modelo SNC-RZ50P de SONY. Son digitales y de resolución máxima de 640x480 píxeles, a una tasa máxima de 25fps con una profundidad de píxel de 24bits y con posibilidad de streaming directo de video MPEG-4. Las cámaras se encuentran instaladas en el hall de la EPS cubriendo la totalidad de la sala tal y como se muestra en la siguiente figura.

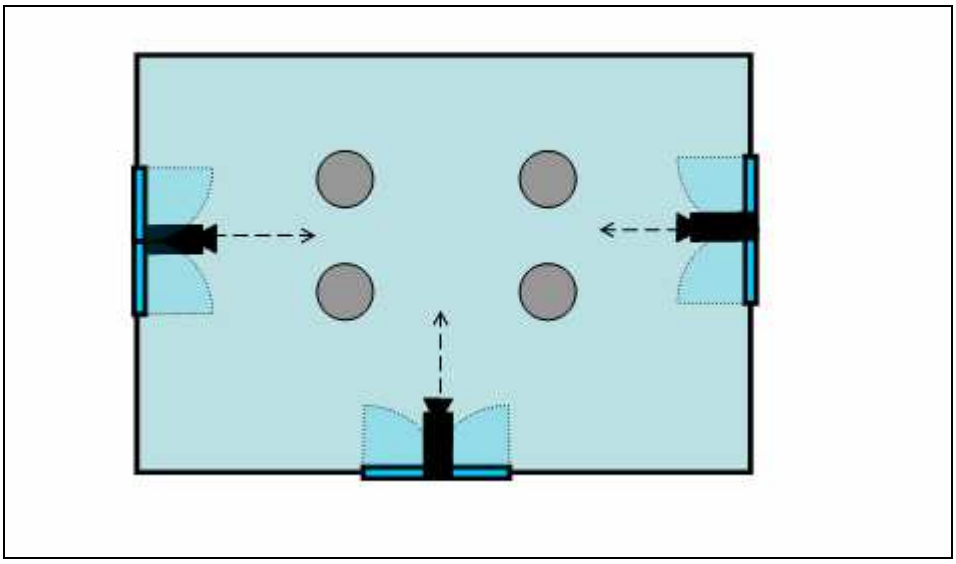


Figura 5-7: cámaras del hall de la escuela

Actualmente para configurar los *frameservers* existen dos maneras: una interfaz web que nos proporciona la cámara PTZ o el fichero de configuración de la aplicación *DiVAFrameServerIP*.

La interfaz web solamente nos permite configurar aspectos propios de la cámara, es decir, formato de captura de datos (calidad de la captura, bpp, transmisión de video,…).
El fichero de configuración no proporciona toda la funcionalidad de las cámaras pero detalla los parámetros básicos de funcionamiento del FrameServer.

Como es lógico, hemos utilizado estas cámaras, porque al encontrarse éstas enfocando a una zona común (aunque puedan moverse y hacer zoom) podemos comprobar el correcto funcionamiento del selector de cuadros.

5.3.2 Ejecución del Sistema de Procesado

En primer lugar, para poder comprobar el correcto funcionamiento del sistema completo, hemos de probar el Sistema de Procesado por separado. Para ello hemos utilizado los *frameservers* comentados anteriormente con la siguiente configuración.

- **Direcciones IP y puertos de las cámaras:** las de las cámaras instaladas en el hall de la EPS de la UAM.
- **Autenticación:** la necesaria para el acceso a las cámaras ya mencionadas.
- **Tamaño de las imágenes:** 320x480 (QVGA). Esta configuración nos permitió obtener mayores tasas de cuadros que el tamaño 640x480 (VGA).
- **Tasa de cuadros por segundo:** la máxima permitida, 25. Dado que la aplicación de procesado tendrá la desventaja de limitar dicha tasa, es conveniente partir de la máxima que nos sea posible para la captura de imágenes.
- **Puerto de servicio:** hemos elegido el 20050, 20051 y 20052, aunque se pueden elegir otros cualquiera a partir del 1024 (que no sean "*well known ports*", que sean efímeros para que no interfieran en la ejecución de otros posibles procesos del PC).

Estos *frameservers* se han ejecutado en la misma máquina que el sistema de procesado, y también de forma independiente en otro PC de la misma subred, y el resultado es aproximadamente el mismo en materia de tasa de cuadros por segundo (en el caso óptimo se obtienen 1 o 2 fps más en la configuración independiente, pero depende siempre del rendimiento la subred que une a ambas máquinas: habitualmente es mejor ejecutar todo en la misma máquina).

A continuación hemos ejecutado la aplicación de procesado (DiVASurveillanceSystem), con la configuración por defecto (y también sin ella, para comprobar que admite la configuración introducida manualmente) y con la opción de visualización de las grabaciones.

En las siguientes imágenes podemos observar los resultados obtenidos, tras muchas pruebas de rendimiento:

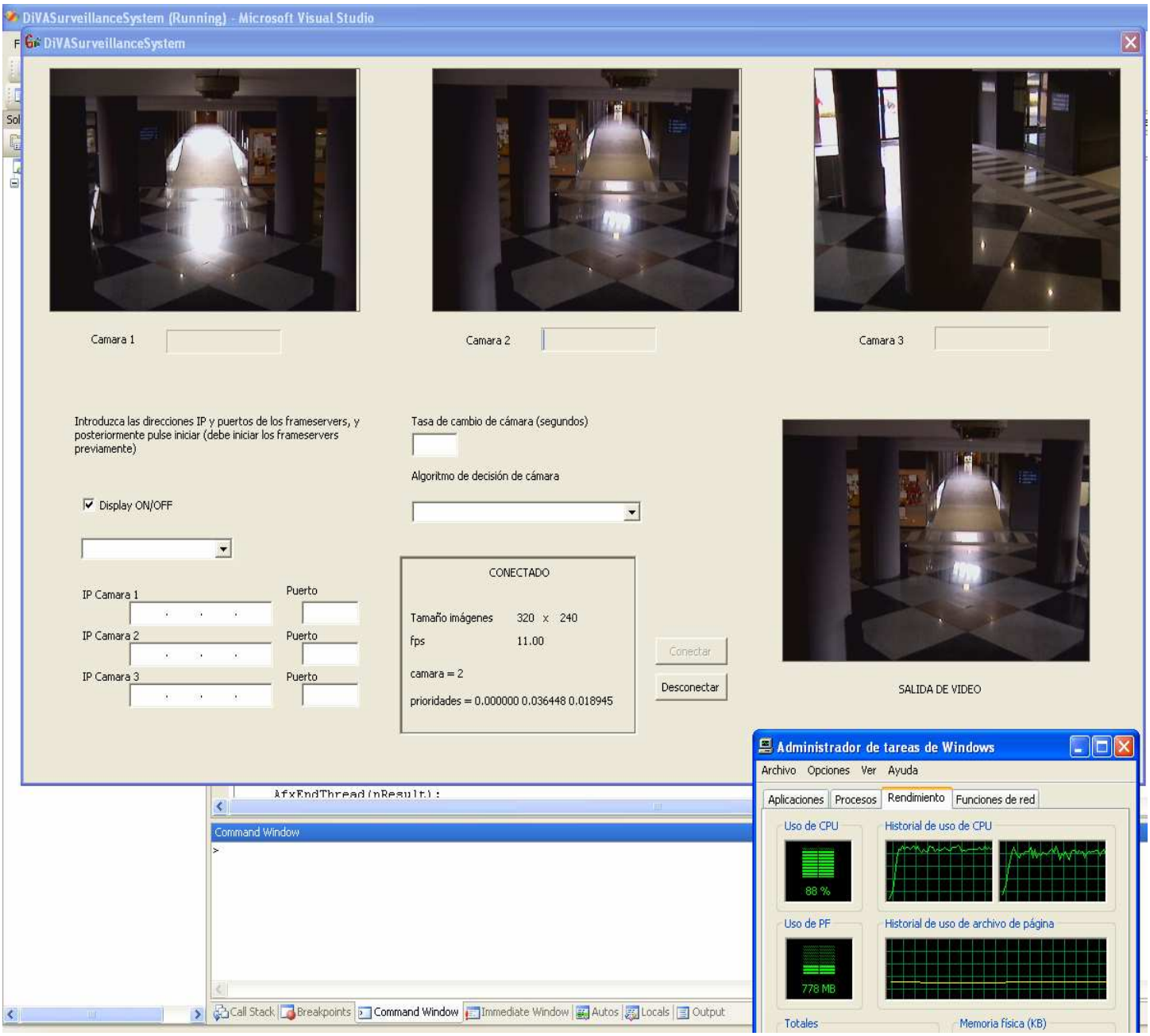


Figura 5-8: captura de pantalla de la interfaz para una escena sin movimiento, con imágenes QVGA

En esta captura (figura 5-8), podemos observar que la tasa de cuadros para imágenes QVGA (320x420) es 11fps, y el rendimiento de la CPU ejecutando la aplicación junto con los frameservers está cerca del 90%. La escena carece de movimiento, cosa que podemos observar tanto en las barras como en el indicador numérico (si no son cero, es por el ruido introducido por las cámaras).

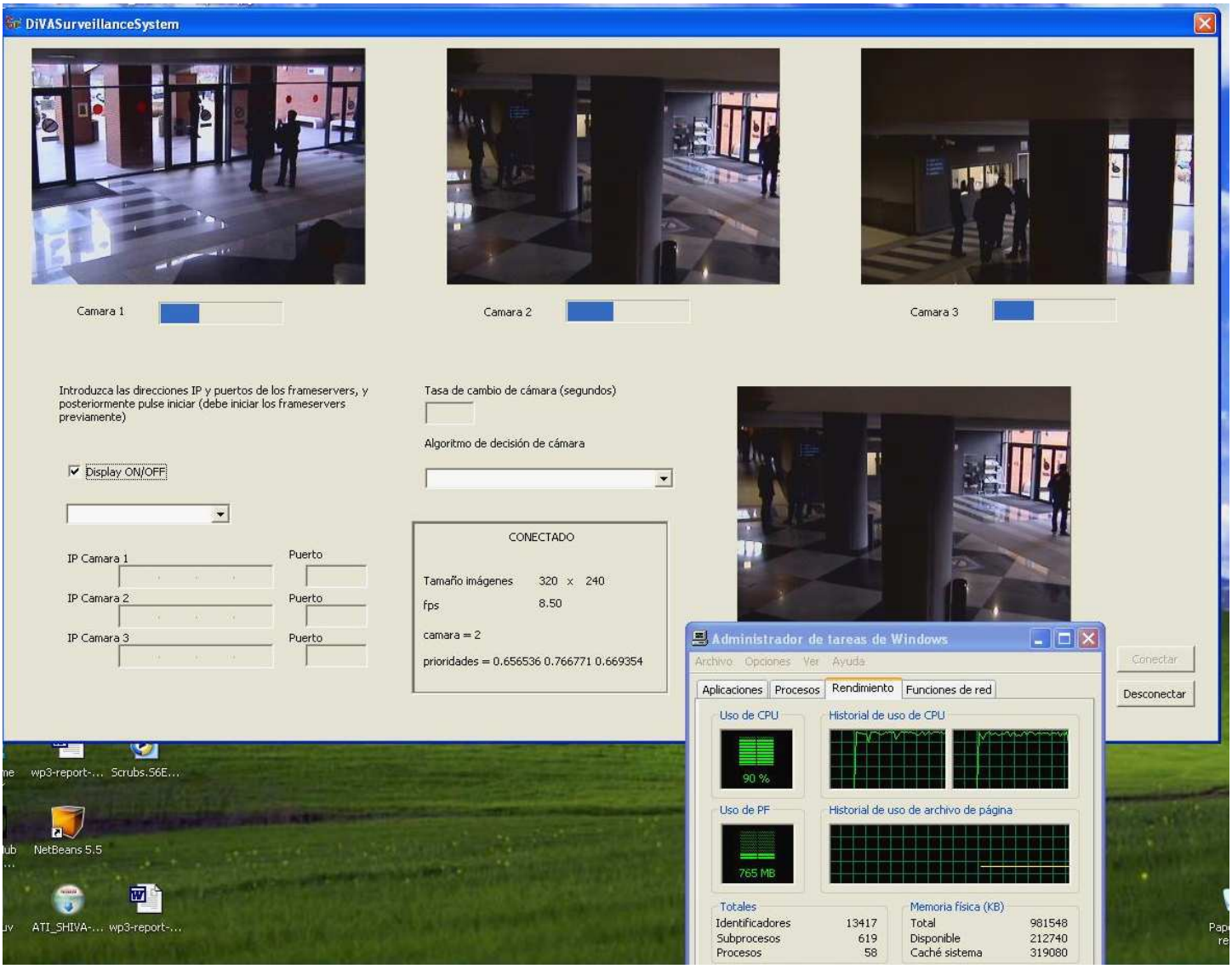


Figura 5-9: captura de pantalla de la interfaz para una escena con movimiento, con imágenes QVGA

Esta captura corresponde al hall de la escuela en una situación con bastante movimiento en la escena (vemos que las barras de las tres imágenes están medio-llenas). La tasa de cuadros obtenida es la máxima posible (12fps) en condiciones de *debug* (en modo *release* pueden obtenerse hasta 15fps, sin las ventanas de video activas), limitada por el Sistema de Procesado y por la red de distribución interna (la conexión con los frameservers).

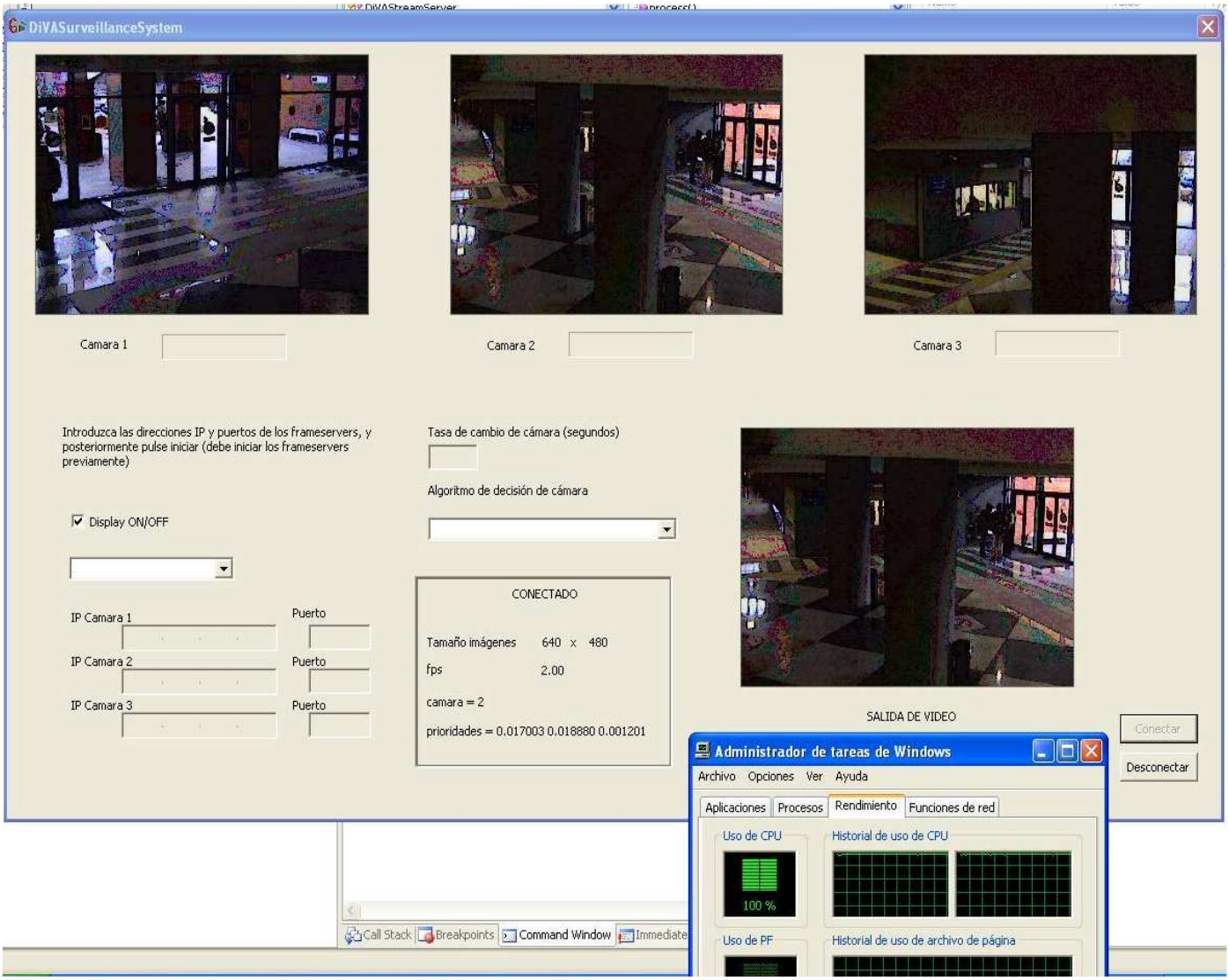


Figura 5-10: captura de pantalla de la interfaz para una escena con poco movimiento, con imágenes VGA

Esta captura corresponde al hall de la escuela en ausencia de movimiento en la escena. La tasa de cuadros obtenida es muy baja (2fps) en condiciones de debug (en release pueden obtenerse hasta 4fps, sin las ventanas de video activas). La limitación de esta tasa puede venir dada tanto por la capacidad de procesado del PC que ejecuta la aplicación (vemos que el uso de CPU cuando solo ejecutamos el Sistema de Procesado es del 100%), como por la velocidad de transmisión de la red interna. Por ello, para las pruebas del apartado siguiente, hemos decidido ejecutar los *frameservers* con un tamaño de imagen QVGA.

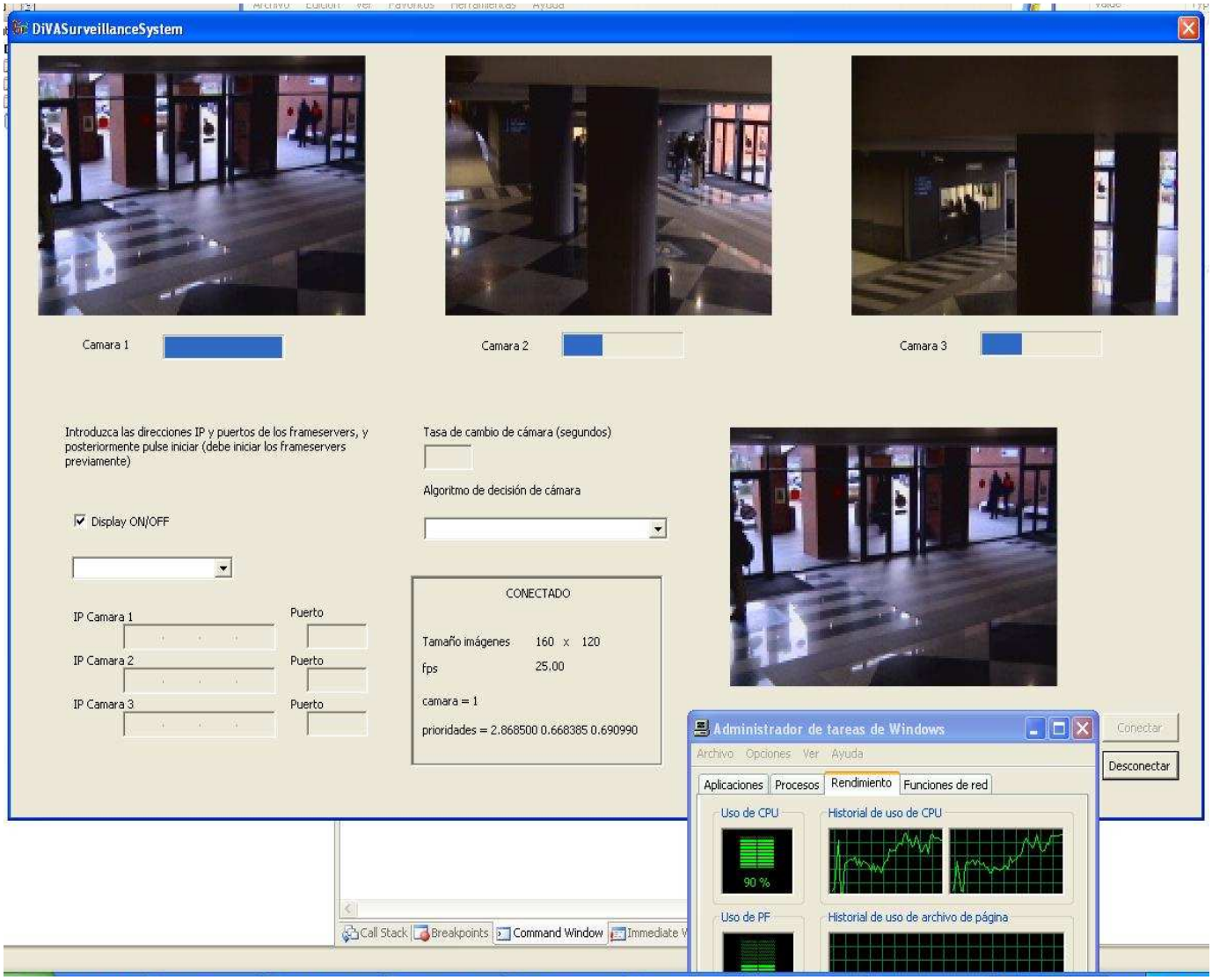


Figura 5-11: captura de pantalla de la interfaz para una escena con bastante movimiento, con imágenes QCIF (160x120)

En esta captura podemos observar la misma escena con bastante movimiento. La tasa de cuadros obtenida la máxima posible que nos ofrecen los frameservers (25fps) aunque si hacemos zoom en la imagen podemos observar que ésta es de mucho peor calidad (debido al pequeño tamaño de ésta). Aunque en la imagen se muestra un rendimiento del 90%, este varía a lo largo del tiempo, pudiendo ser mucho más bajo (como podemos ver en las gráficas temporales del administrador de tareas en la imagen).

Así pues, podemos resumir el rendimiento de la CPU en la siguiente tabla:

Tamaño	Rendimiento (uso de CPU)	Tasa de cuadros por segundo
VGA (640x480)	Cercano al 100%	Menor o igual a 4 fps
QVGA (320x240)	Entre el 80% y 100%	Menor o igual a 15 fps
QCIF (160x120)	Entre el 50% y 90%	Menor o igual a 25 fps

Tabla 5-1: rendimiento del sistema, ejecutando los frameservers y el Sistema de Procesado

Los datos de la CPU donde ha sido ejecutado ambas aplicaciones (*frameservers* y DiVASurveillanceSystem.exe) son:

- Intel® Pentium® 4, 3GHz (2 CPUs)
- 1GB RAM.
- Tarjeta Gráfica RADEONX550 de 256MB de RAM

Estos datos los especificamos para dar a entender que el PC servidor tiene una potencia suficiente, y similar a la de los ordenadores que se comercializan actualmente. Aunque cabría esperar una mejora en el rendimiento si se ejecuta en una máquina más potente (una con mas memoria RAM y 4 Núcleos, por ejemplo), intuimos que esta mejora no será tan acusada por la influencia de la velocidad de la conexión con las cámaras IP (velocidad de la red interna).

5.3.3 Ejecución conjunta con el motor de adaptación CAIN

Hemos visto ya lo relativo al rendimiento del Sistema de Procesado y los servidores de cuadros. Al igual que en la sección anterior, hemos realizado pruebas con los distintos tamaños de imagen posibles, aunque sólo ha sido posible visualizar el video transferido con un tamaño de imágenes: QVGA, 320x240 (actualmente el CAT utilizado no admite mas que un tamaño de imágenes).

El CAT utilizado, como ya vimos en 4.3.2, es el RawVideoCombiner CAT, que se encargaba de producir video MPEG-2 en HTTP desde una fuente de video sumarizado sin compresión. Para ejecutar el motor CAIN con la configuración adecuada hemos:

- Compilado el proyecto RawVideoCombinerTestCase.java, mediante algún entorno de desarrollo JAVA (en este proyecto se ha utilizado netbeans 5.5)
- Configurado adecuadamente el DI raw_tcp_newsitem_di.xml con la dirección de la fuente de video (dirección y puerto donde sirve el DiVAServer de la aplicación de vigilancia).

  Y posteriormente hemos lanzado el ejecutable mediante el comando (ver anexo de instalación):

  ```
  java RawVideoCombinerTestCase
  ```

  En una ventana de “símbolo del sistema”.

De este modo la salida que observamos es la siguiente:

- Lanzamos el ejecutable, y el motor lee el DI. A continuación se toma la decisión y se abre el fichero intermedio.

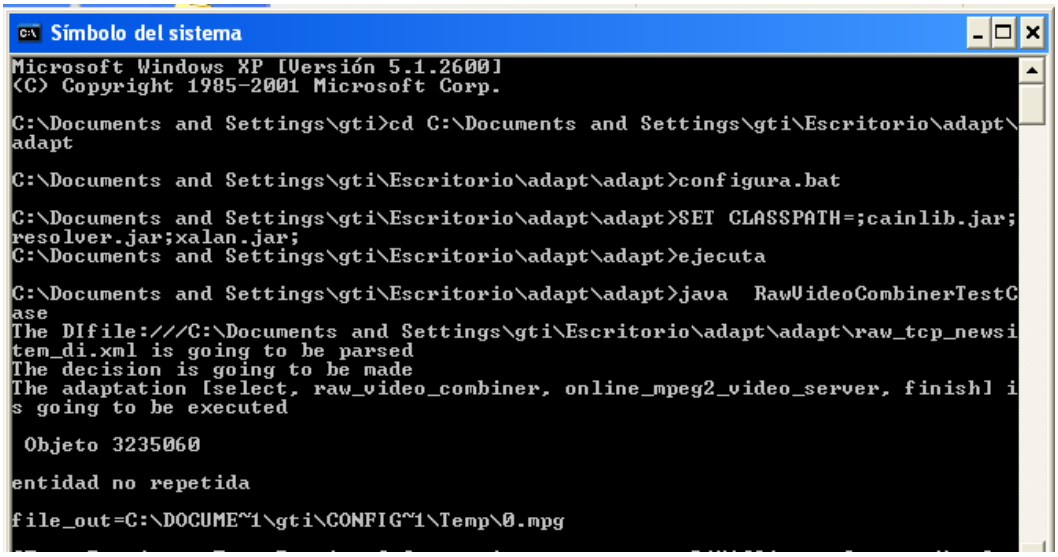


Figura 5-12: ejecución del caso de ejemplo RawVideoCombinerTestCase

- A continuación se conecta al Sistema de Procesado indicado en el DI de configuración y se abre el puerto de servicio de video HTTP (en MPEG-1).

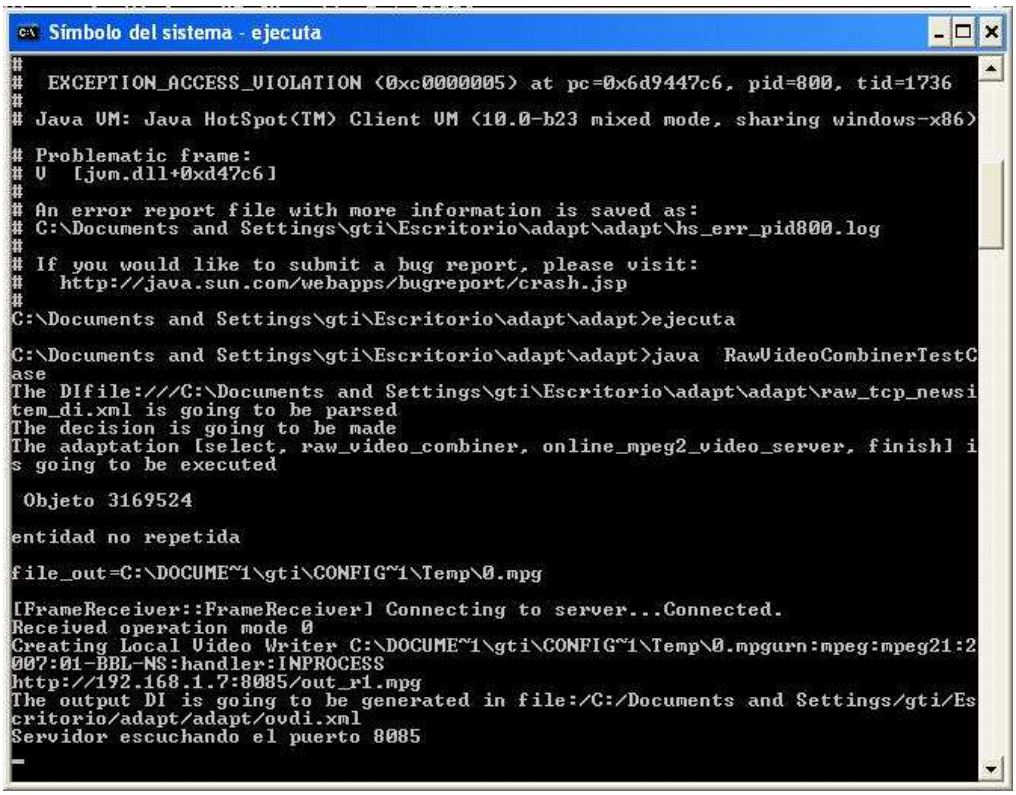


Figura 5-13: conexión con el DiVAServer

- A continuación introducimos en el reproductor utilizado la dirección del video HTTP que nos indica la aplicación.

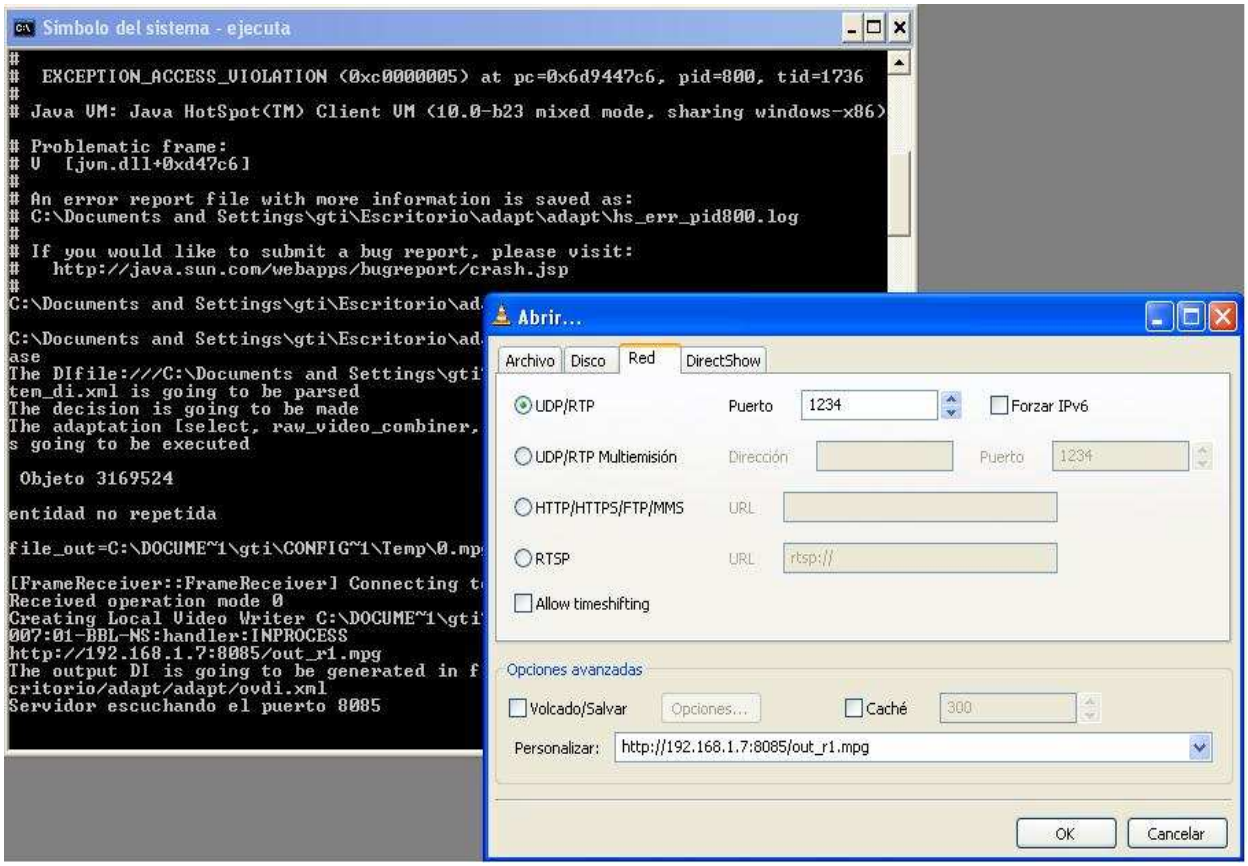


Figura 5-14: Introducir los datos en el reproductor comercial (p.e. VLC media player)

- Conexión del reproductor con CAIN de forma remota. El Motor de CAIN detecta el tipo de terminal que accede, y el software de reproducción del video. El servidor empieza a transmitir y el video se empieza a visualizar en el reproductor.

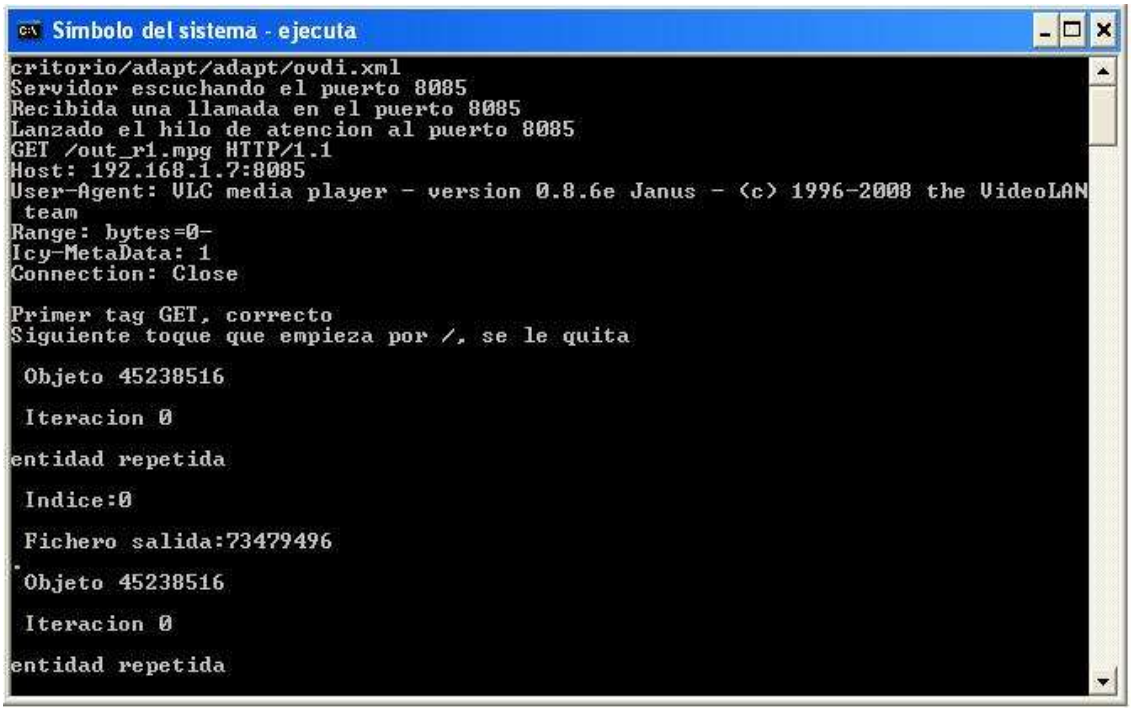


Figura 5-15: comienza la reproducción (VLC en PC)

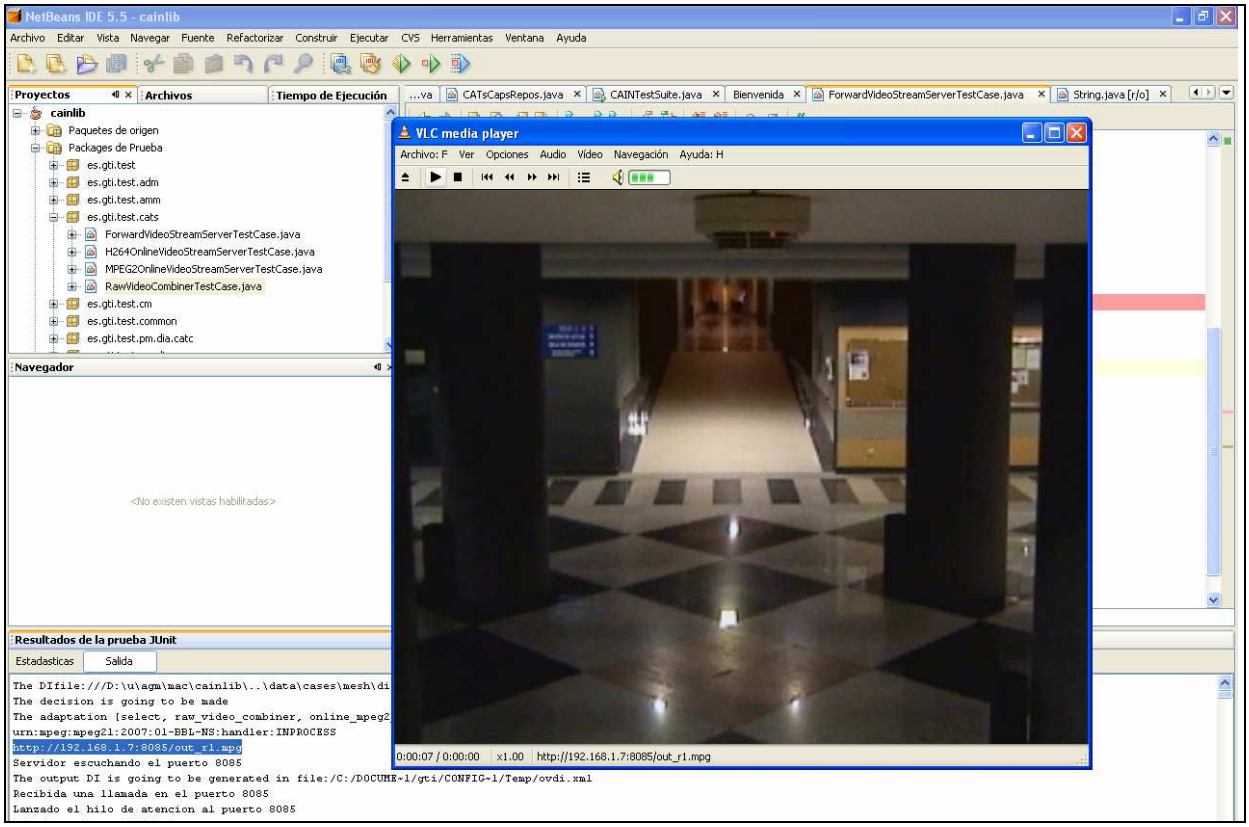


Figura 5-16: captura de pantalla del visualizador en PC (VLC media player)

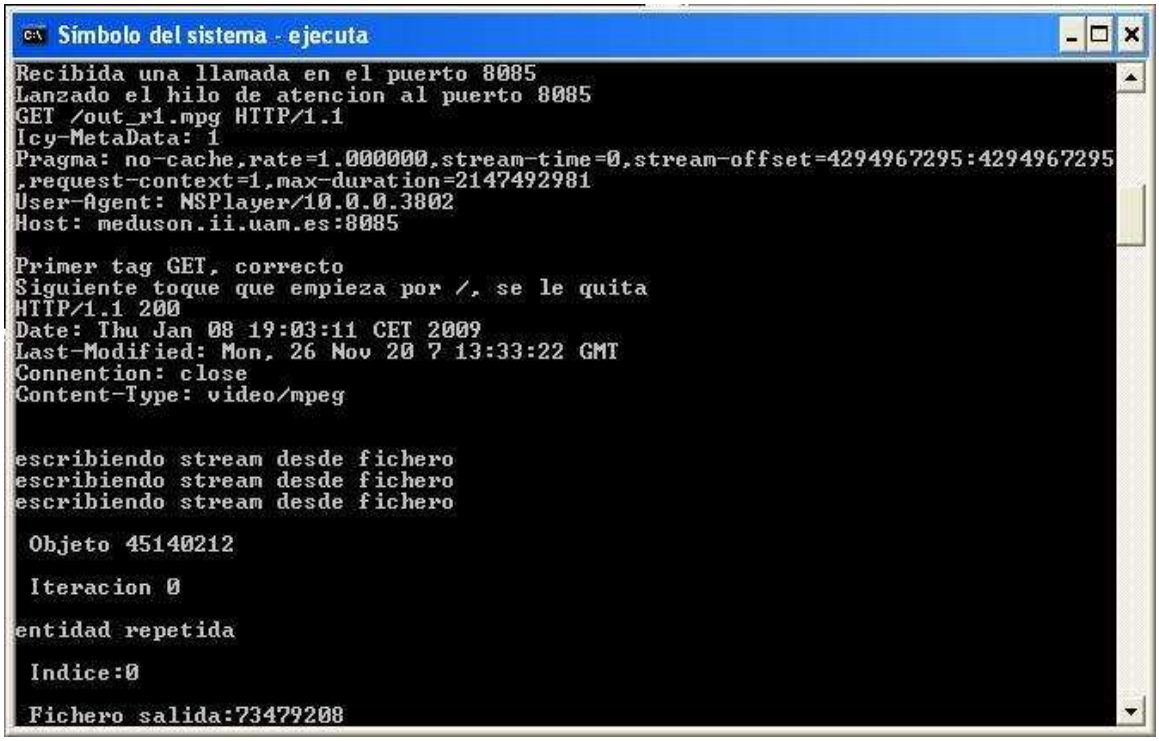


Figura 5-17: comienza la reproducción (TCPMP en PDA)

Figura 5-18: captura de pantalla del visualizador en Windows Mobile (TCPMP)

- Se detiene la reproducción (se pulsa stop)→ se cierra el video y el servidor detiene el envío.

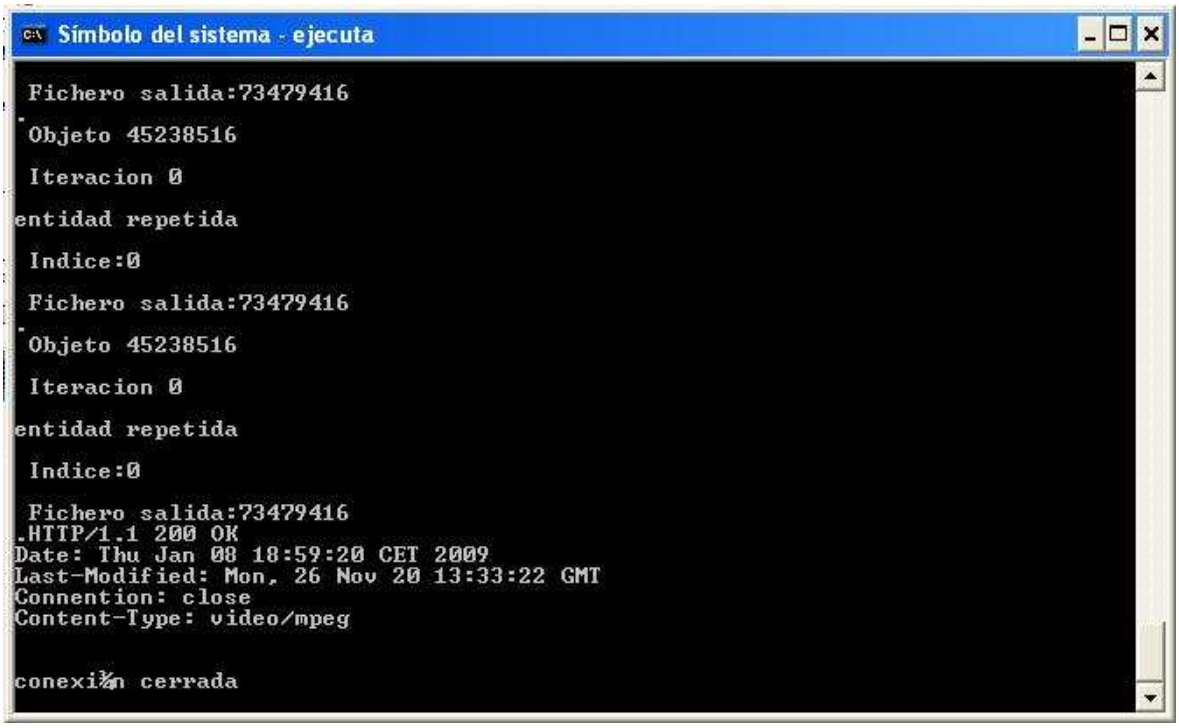


Figura 5-19: se detiene la reproducción

Observando las imágenes reproducidas en la aplicación supervisora (interfaz del Sistema de Procesado) y calculamos el tiempo que transcurre hasta que éstas empiezan a visualizarse en el reproductor remoto, vemos que este asciende a unos 6 segundos. Este retardo, aunque es elevado, tiene su justificación en la propia arquitectura distribuida del sistema. El hecho de tener conexiones entre los *frameservers* con el sistema de procesado, este con el motor de CAIN y este último con la aplicación de visualización introduce de forma implícita un cierto retardo (que además dependerá de la velocidad de transmisión en la red que los une).

Para ver más detalles de la instalación del sistema completo, ver el anexo A.

6 Conclusiones y trabajo futuro

6.1 Conclusiones

Como ya hemos visto en el capítulo anterior, el sistema tiene una funcionalidad muy similar a la deseada, que expusimos en los objetivos del Proyecto, y con un rendimiento "suficiente" para poder desempeñar una labor de videoseguridad.

El primer objetivo era la creación de la aplicación de videoseguridad inteligente, que debía se capaz de monitorizar al mismo tiempo las imágenes captadas por varias cámaras, y procesarlas para seleccionar las más importantes. Esta labor se ha visto realizada en el programa *DiVAMultiSurveillanceSystem*, que se encargaba de obtener las imágenes de los *frameservers*, procesarlas y transmitirlas a la aplicación servidora. Según vimos en la sección 5.3.2, obtuvimos video procesado con una tasa de cuadros de hasta 15 fps, en tamaños de imagen QVGA (tamaño muy utilizado en secuencias de videoseguridad), y con un buen rendimiento del PC que ejecutaba la aplicación.

El segundo objetivo era la creación de la plataforma de streaming del video procesado. Mediante la integración en el sistema de la plataforma de adaptación *CAIN*, y más específicamente, mediante el componente de adaptación *RawVideoCombiner*, conseguimos transformar las imágenes obtenidas de *DiVA* en un video comprimido MPEG-1/2 transmitido mediante el protocolo HTTP. Este video comprimido podía ser visualizado de manera remota mediante un reproductor comercial mediante el codec adecuado.

Aunque el video final tenía una tasa de cuadros aceptable, el retardo obtenido desde la captura hasta la visualización no lo era tanto. Esto se debía a la distribución de los componentes del sistema y a la velocidad de transmisión en la red que interconectaba los subsistemas.

Por tanto, vemos satisfechas las necesidades que en un principio motivaban a la realización de este proyecto fin de carrera.

6.2 Trabajo futuro

En esta sección se incluyen algunos de los aspectos que durante el transcurso y desarrollo de este Proyecto Fin de Carrera, se han considerado como posibles mejoras futuras en el Sistema Inteligente de vídeo-vigilancia.

6.2.1 Respecto al sistema de procesado

- La inclusión e integración de otros algoritmos de procesado en el sistema de procesado, con vistas a crear un sistema más complejo, con mayores posibilidades de detección: algoritmos de seguimiento de objetos, detección de objetos abandonados, etc., así como su posible combinación mediante diversos pesos.

- La evolución de DiVA de forma que admita codificación/decodificación de imágenes (por ejemplo a formato JPEG, JPEG2000, etc). Creemos que una de las limitaciones a la tasa de cuadros por segundo conseguida es la distribución de imágenes en formato raw en el núcleo de red interno. Estas imágenes ocupan un ancho de banda mucho mayor que las imágenes codificadas, con las que probablemente se consigan tasas de cuadros cercanas a los 25fps.

- Inclusión de funcionalidad adicional PTZ en la interfaz de supervisión de la aplicación de videoseguridad. Si un supervisor detecta incidencias quizás querría poder hacer zoom en la cámara que muestra dichas incidencias para ver con mayor precisión la gravedad de éstas.

6.2.2 Respecto al sistema de entrega

- Aprovechar las ventajas que ofrece CAIN a la hora de decidir la adaptación más adecuada en función del cliente que se conecte al sistema. Por ejemplo, si se conectase una PDA, se debe utiliza un determinado descriptor de contexto con propiedades del terminal, de la red, del codecs que admite, etc. distintas a las que tendría el descriptor de contexto de un PC convencional en Windows, Linux o MacOS. Ello requeriría una conexión independiente con el servidor de CAIN, adicional a la de distribución de video HTTP.

6.2.3 Respecto al receptor del video

- Insertar controles adicionales de configuración en el cliente de video, de forma que éste pudiese configurar de forma remota algunos aspectos del sistema, en el caso de no disponer de supervisor. Esto requeriría conexiones adicionales del Sistema de Procesado del cliente y utilizar aplicaciones específicas en vez de reproductores comerciales (significando esto que debemos encontrar o crear nuestro reproductor personalizado). En el caso de los clientes PDA, esta labor sería actualmente muy complicada, tal y como se explica en el Anexo C.3.

Referencias

[1] M. Valera and S.A. Velastin, "Intelligent distributed surveillance systems: A Review", IEEE Procs. of VISP, 152(2):192-204, 2005

[2] A. D. Bue, D. Comaniciu, V. Ramesh, and C. Regazzoni, "Smart cameras with real-time video object generation," in Proc. IEEE Intl. Conf. on Image Processing, Rochester, NY, volume III, 2002, pp. 429–432.

[3] K.N. Platanioitis, C.S. Regazzoni (eds.), "Special Issue in Visual-centric Surveillance Networks and Services", IEEE Signal Processing Magazine, 22(2), Marzo 2005.

[4] G. R. Bradski, "Computer vision face tracking as a component of a perceptual user interface," en Proc.IEEE Workshop on Applications of Computer Vision, Princeton, NJ, October 1998, pp. 214–219.

[5] A. Hampapur, L. Brown, J. Connell, A. Ekin, N. Haas, M. Lu, H. Merkl, S. Pankanti, A. Senior, C. Shu, and Y. Li Tian "Detection and Tracking in the IBM PeopleVision System", IEEE SIGNAL PROCESSING MAGAZINE MARCH 2005

[6] A. Albiol, C. Sandoval, A. Albiol, "Robust Motion Detector for Video Surveillance Applications". Proc. of the International Conference on Image Processing. Barcelona. Spain. 2003.

[7] Lee, J. Y. B.Sons, J. W. &., ed. (2007), "Scalable continuous media streaming systems".

[8] J.C. San Miguel, J. Bescós, J.M. Martínez and A. García, "DiVA: a Distributed Video Análisis framework applied to video-surveillance systems", Ninth International Workshop on Image Analysis for Multimedia Interactive Services (...). . In: Proceedings of the WIAMIS'08 (2008)

[9] J.M. Martínez, V. Valdés, J. Bescós, L. Herranz, "Introducing CAIN: a Metadata-driven Content Adaptation Manager Integrating Hetereogeneous Content Adaptation Tools" . In: Proceedings of the WIAMIS'2005 (2005)

[10] A. García, J. Molina, J.C. San Miguel, J. Bescós, J.M. Martínez, V. Fernánandez. "Documentación del sistema DiVA: Distributed Video Analysis Framework" (2006-2008)

[11] V. Valdés, J.M. Martínez, "Content Adaptation Tools in the CAIN framework", in Visual Content Processing and Representation, L. Atzori, D.D. Giusto, R. Leonardi, F. Pereira (eds.), Lecture Notes in Computer Science, Vol. 3893, Springer Verlag, 2006, pp. 9-15

[12] V. Valdés, J.M. Martínez, "Content Adaptation Capabilities Description Tool for Supporting Extensibility in the CAIN Framework", Multimedia Content Representation, Classification and Security-MCRS2006, B.Günsel, A.K.Jain, A.M. Tekalp, B. Sankur (eds.), Lecture Notes in Computer Science, Vol. 4105, Springer Verlag, 2006, pp. 395-402

[13] F. López, J.M. Martínez, V. Valdés, "Multimedia Content Adaptation within the CAIN framework via Constraints Satisfaction and Optimization", Proceedings of the Fourth International Workshop on Adaptative Multimedia Retrieval-AMR06 (CD-ROM), 17 pp., Geneve, Suiza, July 2006.

[14] ISO/IEC 15938-5, Information Technology – Multimedia Content Description Interface – MPEG-7 Part 5: Multimedia Description Schemes

[15] ISO/IEC 21000-7, Information Technology – Multimedia Framework – part 7: Digital Item Adaptation
[16] V. Ferrari, T. Tuytelaars, L. Van Gool, “Simultaneous Object Recognition and Segmentation by Image Exploration”, Computer Vision Group (BIWI), ETH Zürich, Switzerland, ESAT-PSI, University of Leuven, Belgium
[17] http://ffmpeg.mplayerhq.hu/ffmpeg-doc.html

Glosario

3G	3rd Generation
3GPP	3rd Generation Partnership Project
ADM	Adaptation Decision Module
ADM	Adaptation Decision Module
AEM	Adaptation Execution Module
AMM	Adaptation Management Module
API	Aplication Programming Interface
ARM	Advanced RISC Machine
AVI	Audio Video Interleaved
BSD	Berkeley Software Distribution
CAIN	Content Adaptaptation INtegrator
CAT	Content Adaptation Tool
CCD	Charged Coupled Device
CCTV	Circuitos Cerrados de TeleVisión
CDK	Content Developing Kit
CIF	Common Intermediate Format
CM	Coupling Module
CME	Cain Module Extensibility
COM	Component Object Model
CPU	Central Pocessing Unit
DI	Digital Item
DIA	Digital Item Adaptation
DiVA	Distributed Video Analysis
DLL	Dynamically Linked Library
DM	Decisión Module
DRM	Digital Rights Management
DSS	Darwin Streaming Server
FPS	Frames Per Second
GigE	Gigabit Ethernet
GPS	Global Positioning System
GSM	Global System for Mobile communications
GUI	Graphical User Interface
HSDPA	High-Speed Downlink Packet Access
HTML	HyperText Markup Language
HTTP	Hypertext Transfer Protocol
HW	Hardware
IrDA	Infrared Data Association
JPEG	Joint Photographic Experts Group
MDS	Multimedia Description Schemes
MFC	Microsoft Fundation Classes
MIL	Matrox Imaging Library
MMS	Microsoft Media Server protocol
MP3	MPEG-1 Audio Layer 3
MPEG	Moving Picture Experts Group
OpenCV	Open Source Computer Vision
OS	Operating System

PDA	Personal Digital Assistant
PM	Parsing Module
PTZ	Pan Tilt Zoom
QCIF	Quarter Common Intermediate Format
QTSS	Quick Time Streaming Server
QVGA	Quarter Video Graphics Array
RTCP	Real-time Transport protocol Control Protocol
RTP	Real-time Transport Protocol
RTSP	Real Time Streaming Protocol
SDK	Software Developing Kit
SGBD	Sistema de Gestión de Bases de Datos
SP	Service Pack
UED	Usage Environment Description
UMA	Universal Multimedia Access
URL	Uniform Resource Locator
USB	Universal Serial BUS
VGA	Video Graphics Array
VoD	Video On Demand
VPULab	Video Processing and Understanding Lab
WLAN	Wireles Local Area Network
WM/WMP	Windows Media/Windows Media Player

Anexos

A Manual de instalación

A continuación expondremos las pautas generales para instalar el sistema distribuido.

A.1 Instalación general

En primer lugar volvemos a recordar la división en módulos del sistema completo:

- Servidor de cuadros (frameserver)
- Sistema de Procesado
- Servidor de video
- Cliente reproductor de video

La configuración que elegimos para realizar las pruebas en este Proyecto Fin de Carrera (distribución de los módulos), recordamos, es la siguiente:

- PC #1: Servidores de cuadros + Sistema de procesado (aunque pueden separarse estos módulos en distintas máquinas, con las ventajas e inconvenientes que vimos en anteriores capítulos. Llamaremos esta máquina la **Procesadora de video**.
- PC #2: Servidor de Video Procesado. Llamaremos a este PC por tanto el **Servidor de Video**.
- PC #3 ó dispositivo PocketPC: **Cliente reproductor de video**.

Explicaremos ahora con detalle los requisitos software para la ejecución de los módulos que componen este sistema. Recordamos que en los dos primeros PCs debe estar instalado el sistema operativo Windows® XP o Vista para que funcionen los módulos.

A.1.1 Procesador de Video

Esta máquina, que ejecutará programas basados en la plataforma de DiVA, requiere (recordamos) los siguientes paquetes software:

- **Microsoft Fundation Classes (MFC)** 6.0. La mejor forma de obtener esta librería es instalar el entorno de desarrollo Microsoft® Visual C++ (a partir de la versión 6) o Microsoft® Visual Studio (este proyecto se ha desarrollado en la versión 2005, aunque hay versiones más actuales).

- **Intel® Open Source Computer Vision Library** versión 1.0 (OpenCV). El *core* de la plataforma DiVA esta desarrollado sobre el core de OpenCV. Esta librería es de código abierto, y por tanto puede obtenerse descargándolo de la red.

En primer lugar, vamos a necesitar para compilar los ejecutables del sistema la librería de DiVA actualizada (puede obtenerse desde el repositorio del servidor del VPULab) y compilada en el ordenador en el que queramos ejecutar los programas basados en DiVA (esto es muy importante, pues si utilizamos la librería compilada en otro PC, aunque tenga unas características similares, pueden darse fallos al compilar los programas). El nombre de la librería es **LibDiVABasic**, compilada desde el proyecto con el mismo nombre (en *Debug* o *Release* según deseemos depurar o ejecutar con el máximo rendimiento posible).

Para la ejecución de los frameserversIP es necesario compilar el proyecto DiVAFrameservers, que produce los ejecutables de los servidores de cuadros de distintas fuentes. A nosotros nos interesa el **DiVAFrameServerIP,** y para su ejecución debemos acompañarlo de los siguientes ficheros (tal y como podemos ver en la siguiente figura):

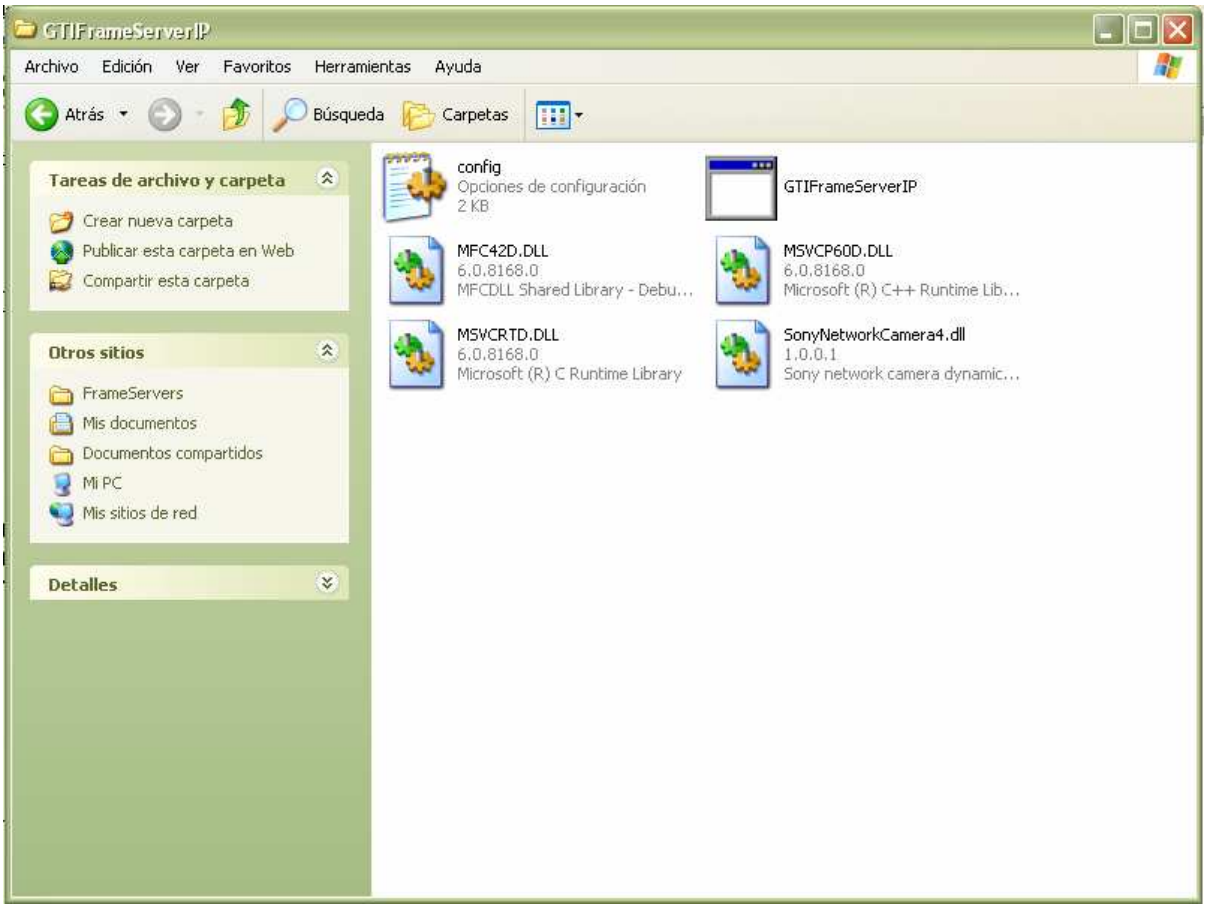


Figura A-0-1: contenido de la carpeta con el ejecutable del Frameserver

Como vemos, son necesarias librerías de Visual Studio (pueden encontrarse donde se encuentra instalada el entorno de desarrollo) y la librería de la cámara utilizada (en este

caso la SONY SNC-RZ50P). En la siguiente subsección A.1.1.1 podemos ver la estructura del fichero de configuración (config).

En cuanto al ejecutable del Sistema de Procesado, **DiVAMultiSurveillanceSystem**, es preciso compilar el proyecto de mismo nombre (debe incluirse la carpeta del proyecto en el mismo nivel donde se encuentran los proyectos de procesado de diva: "PROCESSING_SYSTEM→PROYECTS"), el cual solo precisa la librería DiVA (LibDiVABasic) y las fuentes de código del proyecto ICSS (también puede descargarse del repositorio del VPULab) en el mismo nivel de carpetas que las fuentes de DiVA. La carpeta del proyecto incluye una serie de librerías dinámicas externas a DiVA necesarias para la ejecución del Sistema de Procesado.

Una vez obtenidos los dos ejecutables, podemos lanzar el sistema conjunto de esta forma:

- Primero lanzaremos los frameservers con la configuración de las cámaras que queramos usar.
- A continuación una vez lanzado la aplicación **DiVAMultiSurveillanceSystem** (actualmente compilada para su uso con 3 cámaras), podemos pulsar en Conectar, y se iniciará el Sistema de Procesado. En ese momento, podríamos lanzar la aplicación de CAIN en el Servidor de Video.
- Para detener la ejecución pulsar en Desconectar.

A.1.1.1 Fichero de configuración de los frameservers

A continuación se muestra el aspecto de un fichero de configuración típico de dicho *FrameServer*. Respecto a los modos de configuración, todas las combinaciones son virtualmente posibles aunque se tendrán que tener en cuenta aspecto como el consumo de ancho de banda. En el capítulo de Pruebas y Resultados hablamos de la configuración utilizada en las pruebas de ejecución del sistema.

```
// ARCHIVO DE CONFIGURACIÓN DEL FRAME SERVER IP
// Admite comentarios, lineas en blanco y lineas con parejas nombre -
valor (separados por un signo '=')

////////////////////////////////////
// Parámetros de la capturadora IP //
////////////////////////////////////
// Dirección IP de la cámara
// CAM_IP_ADDR=192.168.1.100
CAM_IP_ADDR=192.168.1.101
// CAM_IP_ADDR=192.168.1.102

////////////////////////////////////
// Puerto de la cámara
CAM_PORT=80

////////////////////////////////////
// Proxy
```

```
PROXY_FUNC=0

////////////////////////////////////
// Autenticación
AUTHENTICATION_FUNC=1
USER=admin
PASS=admin

////////////////////////////////////
////////////////////////////////////
// Tamaño de las imágenes
//FRAME_SIZE_X=640
//FRAME_SIZE_Y=480

FRAME_SIZE_X=320
FRAME_SIZE_Y=240

// FRAME_SIZE_X=160
// FRAME_SIZE_Y=120

////////////////////////////////////
// Cuadros por segundo
//FPS=25
// FPS=20
FPS=16
// FPS=12
// FPS=8
// FPS=6
// FPS=5
// FPS=4
// FPS=3
// FPS=2
// FPS=1

////////////////////////////////////
// Formato de captura
COLOR=color
//COLOR=black

////////////////////////////
// Parámetros del servidor //
////////////////////////////
// Puerto de servicio
PORT=20052
```

A.1.2 Servidor de Video

Para la instalación del Servidor de Video, necesitamos tener descargado en el PC correspondiente el proyecto MAC (disponible en el repositorio del VPULab), que contiene las fuentes para la compilación del motor de CAIN en Java/C++. En primer lugar, es necesario compilar la librería de CAIN, llamada CainLib.jar.

Para poder ejecutar el programa es necesario tener instalado algún entorno de desarrollo de Java. A continuación es necesario copiar el fichero ***java.exe.manifest*** en la misma carpeta donde se halla el ejecutable de Java (probablemente en Archivos de

programa\Java\bin), para que CAIN pueda cargar las librerías de DiVA y las de FFMPEG, necesarias para que el CAT *RawVideoCombiner* funcione.

También es necesario instalar Visual Studio con su Service Pack 2. De otro modo, durante la ejecución del programa tendría lugar un error al cargar las DLLs de DiVA.

Es posible ejecutar el programa desde un entorno de desarrollo Java, compilando el CAT RawVideoCombiner, en el *namespace* es.gti.test.cats (en este proyecto se ha utilizado netbeans). No obstante, esta combinación daría una ejecución más lenta y de menor rendimiento que de la siguiente forma.

La otra opción, es llamar al ejecutable desde consola (mediante *scripts*), desde la carpeta adapt, para ejecutar el programa sin necesidad de entorno de desarrollo:

- En primer lugar ejecutamos el *script* "configura.bat", que lo que hace es establecer en el PATH (ruta del sistema) la ruta de las librerías Java que se incluyen en la carpeta.
- A continuación lanzamos "ejecuta.bat" que lo que hace es una llamada a "java RawVideoCombinerTestCase", que como podemos ver ejecuta el CAT ya compilado.

Esta opción requiere la carpeta "adapt", que contiene los archivos que vemos en la siguiente figura:

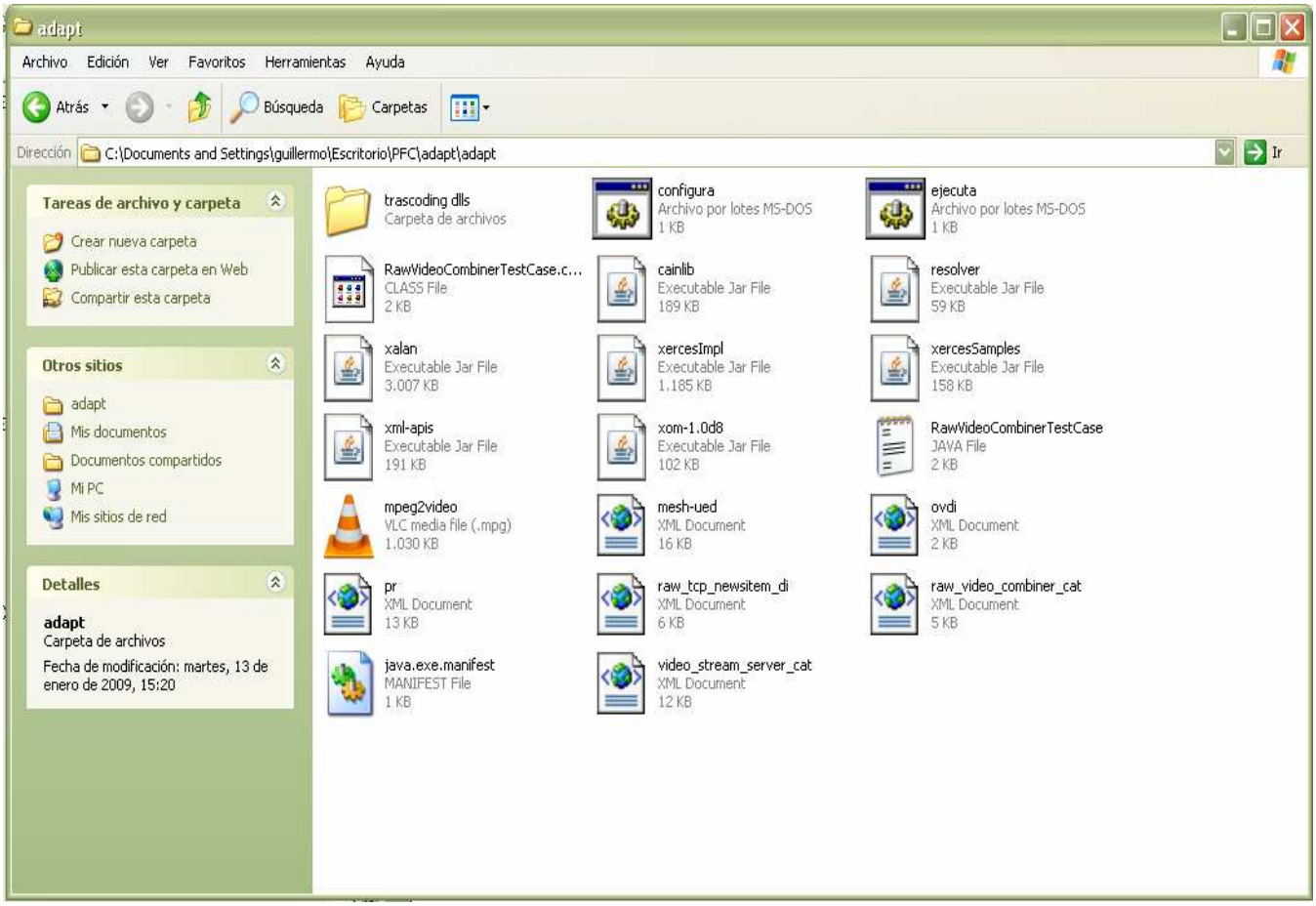


Figura A-0-2: contenido de la carpeta adapt

Por tanto, esta carpeta contendría:

- Los scripts
- Los DI (Digital Ítems), ficheros de propiedades y los CATs (ficheros .xml).
- Las librerías Java.
- Una carpeta con las librerías en C++ (transcoding DLLs).
- La clase Java del CAT (RawVideoCombinerTestCase)
- El video de referencia del CAT
- El *manifest* de java.exe

Para poder obtener la funcionalidad deseada hemos de modificar el DI "raw_tcp_newsitem_di.xml", referenciando a la máquina (IP y puerto) que ejecuta el Sistema de Procesado. En el Anexo D.1 podemos ver la estructura del DI, y resaltado el lugar donde configurar esto (sobre fondo amarillo).

A.1.3 Cliente Reproductor de video

Para la visualización del video procesado, de forma remota, es necesario instalar en el PC o PDA la aplicación que nos proporcione esta funcionalidad, que en este caso es un reproductor conocido.

Las pruebas de este Proyecto Fin de Carrera han sido realizadas con el reproductor del proyecto VideoLAN (VLC media player), aunque podrían utilizarse otros reproductores que admitan *streamings* de video MPEG-1/2 con el protocolo HTTP (por ejemplo Windows Media Player con el codec apropiado).

En la versión para PC, puede descargarse desde la ubicación http://www.videolan.org/vlc/ el instalador de este reproductor (ya vimos que era un proyecto software de código abierto). Como vimos en la sección Pruebas y Resultados, hemos de introducir la dirección del streaming que queremos reproducir en el menú "Medio→Red→Dirección", como vemos en la siguiente figura:

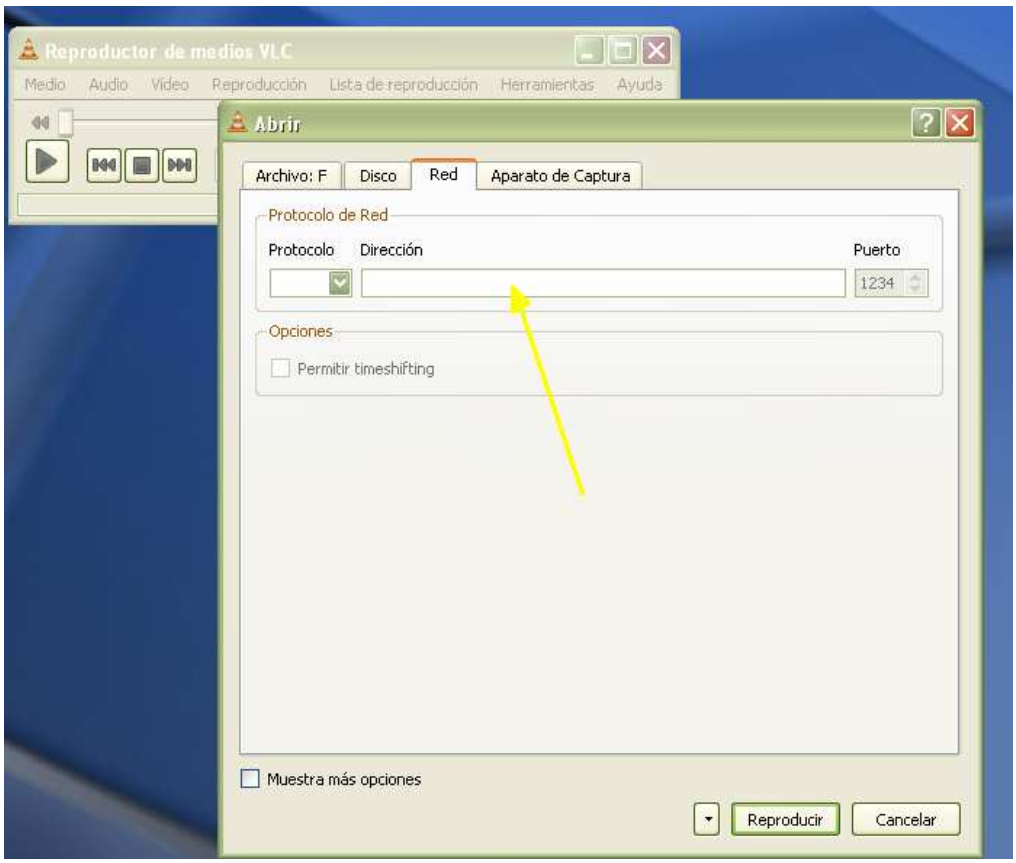


Figura A-0-3: configuración de VLC (PC)

El caso de la PDA es algo más complicado. Como veremos en el siguiente Anexo C.2.5, compilar nuevas versiones del reproductor para Windows Mobile es una tarea muy compleja. Lo mejor es buscar en Internet alguna versión antigua ya compilada, y probar que soporte el streaming de nuestro sistema. Al igual que en PC, debemos introducir en el menú "file→open network stream" la dirección y puerto del Servidor de Video Procesado.

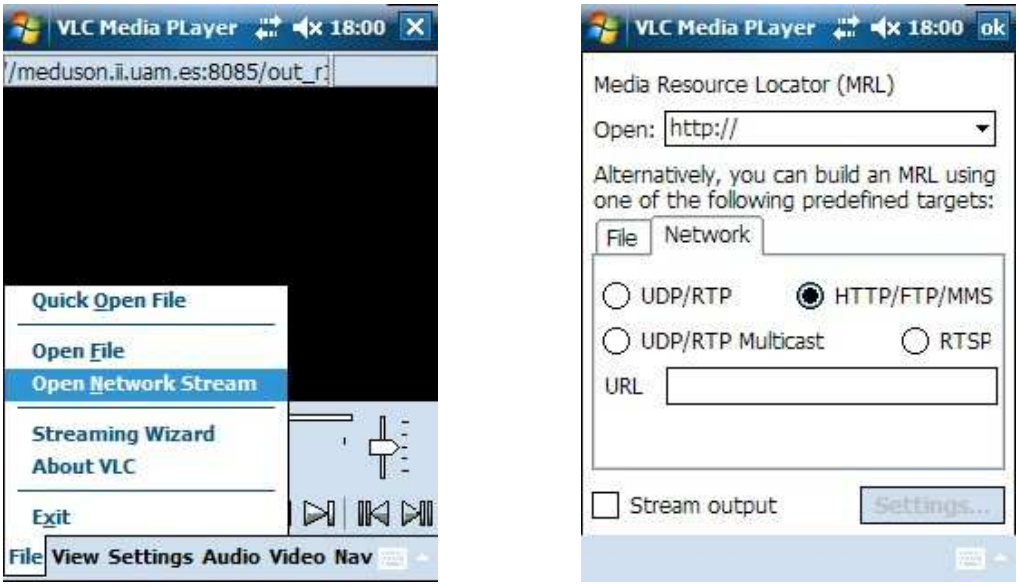


Figura A-0-4: Introducir URL en VLC Media Player

En el proyecto hemos utilizado además otro reproductor que nos permitía también visualizar este video procesado, el TCPMP (ver Anexo C.2.3). Este reproductor puede ser también descargado de Internet en la web http://www.hpcfactor.com/downloads/tcpmp/ (en ella podemos encontrar no solo el ejecutable, sino las fuentes de varias versiones y códecs adicionales). Para reproducir el contenido del streaming, abrimos el menú "Archivo→Abrir Archivo".

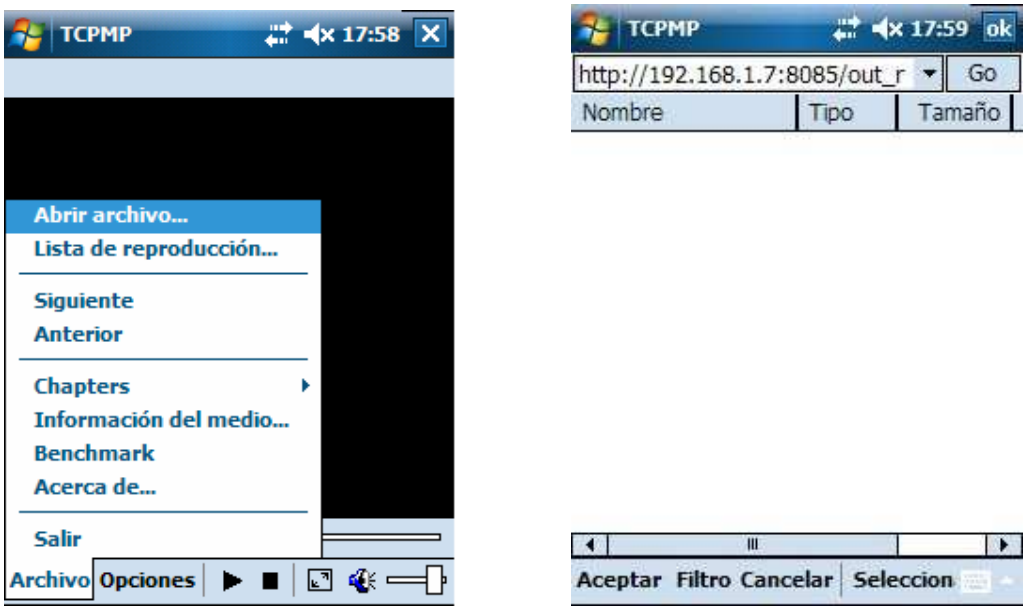


Figura A-0-5: Introducir URL en TCPMP

B DiVA: Distributed Video Análisis

B.2 Arquitectura de la red de distribución

La red de distribución es la encargada de interconectar los distintos sistemas que componen la plataforma DIVA. Los principales objetivos de la red de distribución son los siguientes:

- Distribuir el contenido adquirido por el sistema de captura por toda la red
- Distribuir el contenido de las bases de datos por toda la red
- Proporcionar interfaces entre los distintos subsistemas que componen la plataforma

En el siguiente cuadro se resumen las tecnologías usadas para la diseñar la red de distribución de la plataforma DiVA.

Capas Modelo OSI	Tecnología Utilizadas
Capa transporte	TCP
Capa red	IP
Capa enlace	Ethernet
Capa física	Par Trenzado

Tabla B-0-1: Tecnologías seleccionadas para la red de distribución

Se ha elegido el uso de estas tecnologías debido a su versatilidad y a su facilidad de uso en diferentes entornos.

La red de distribución se compone a su vez de dos subredes: un núcleo central de alta velocidad y una red exterior de velocidad media.

- El **núcleo central** incluye el conjunto de equipos que debido al diseño del sistema necesitan un mayor uso de recursos de red (como son los subsistemas de captura de datos y de almacenamiento en bases de datos). Este sistema consta de varios equipos conectados entre si mediante tecnología Gigabit Ethernet. Además se incluyen todas las conexiones necesarias para la captura de vídeo desde las distintas fuentes de vídeo de la plataforma. Este núcleo de red se conecta con la red exterior mediante un router Gigabit.
- La **red exterior** incluye todos los algoritmos de procesado. Los equipos que ejecutan algoritmos requieren un menor ancho de banda (debido principalmente a que no tienen que realizar múltiples conexiones) y este hecho posibilita su uso en redes de peor calidad. Actualmente la tecnología de dicha subred es de Ethernet a 100Mbps.

Para realizar la interconexión entre los distintos subsistemas se utilizan sockets TCP/IP de Windows. Dichos sockets han sido implementados usando la librería MFC de Microsoft.

El protocolo de conexión se ha simplificado lo máximo posible. Sin embargo esta simplificación no restringe que se pueda complicar el protocolo de comunicación añadiendo mecanismos de autentificación segura y otros.

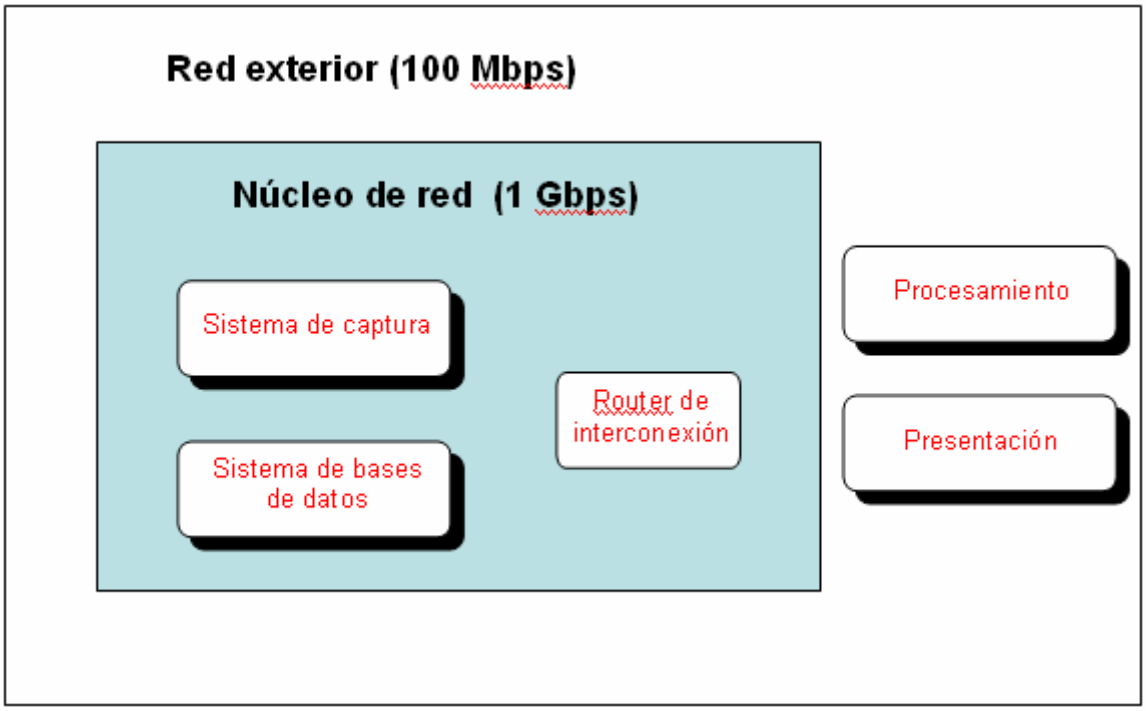


Figura B-0-6: Interconexión de los sistemas de DiVA

Antes de realizar la interconexión para realizar la transferencia de datos, el sistema cliente realiza los siguientes pasos:

- Conexión al sistema servidor por un determinado puerto
- Solicitud de una identificación (ID) de cliente para realizar la transferencia de datos
- El servidor envía el número ID al cliente
- Comienza el intercambio de datos

Por cada petición de datos del cliente, el servidor lanza un hilo que atiende dicha petición (con lo cual se permite la interacción de varios sistemas cliente con un servidor

B.3 Arquitectura del sistema de captura

Este sistema es el responsable de la adquisición de la señal de vídeo de las fuentes de la plataforma, la adaptación al formato usado en la plataforma y la distribución de los cuadros (*frames*) de la señal de vídeo a los algoritmos de procesamiento.

Para cada fuente de captación de señal de vídeo se ha diseñado una interfaz para poder adaptar los datos que proporcionan las fuentes a los datos manejados por la plataforma.

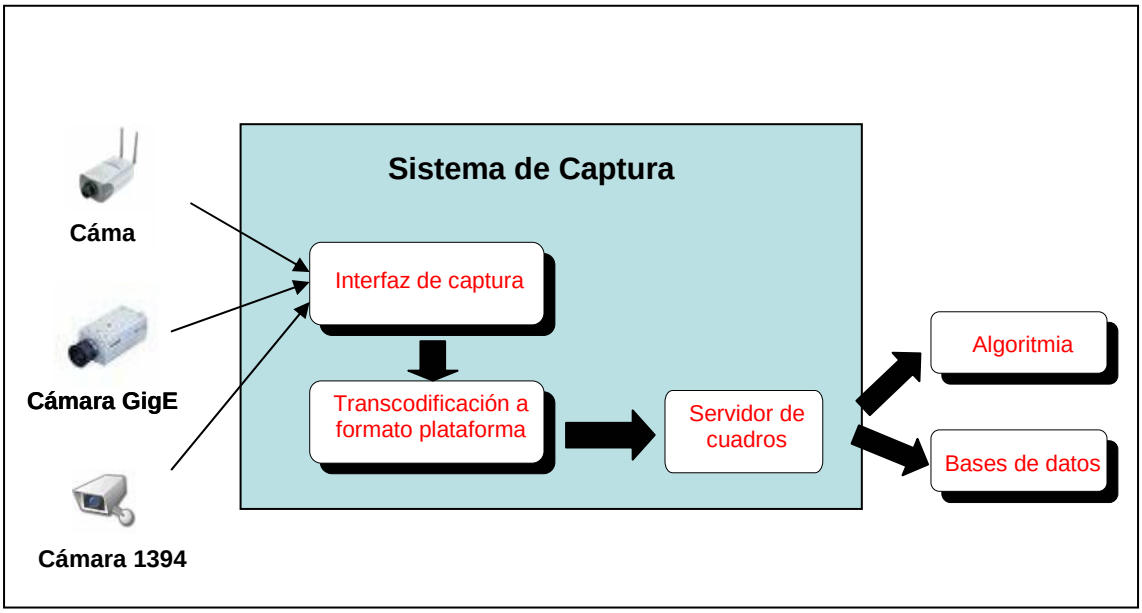


Figura B-0-7: Sistema de captura de señal de vídeo de la plataforma DiVA

Actualmente existe un sistema de captura por cada dispositivo conectado a la plataforma DiVA. Cada sistema se ha nombrado con la palabra "*FrameServer*" seguido de algún identificador que representa la fuente de vídeo (*FrameServer1394, FrameServerPTZ, FrameServerGigE, FrameServerFile,…*).

B.4 Arquitectura del sistema de base de datos

El sistema de bases de datos de la plataforma DiVA maneja distintos tipos de información. Actualmente el sistema maneja tanto información proveniente del análisis de datos como información de contexto de aplicación. Para poder utilizar de una manera eficiente ambos tipos de información se han diseñado dos tipos diferentes de bases de datos:

- Una base de datos para almacenar resultados del análisis de los algoritmos
- Una base de datos para almacenar descripciones de contexto de aplicación, modelos de objetos, parámetros de algoritmos,…en resumen almacena conocimiento que puede ser usado tanto a priori como a posteriori por los algoritmos de análisis.
- A continuación en la siguiente figura se muestra un esquema del flujo de datos entre las distintas bases de datos y los algoritmos del sistema.

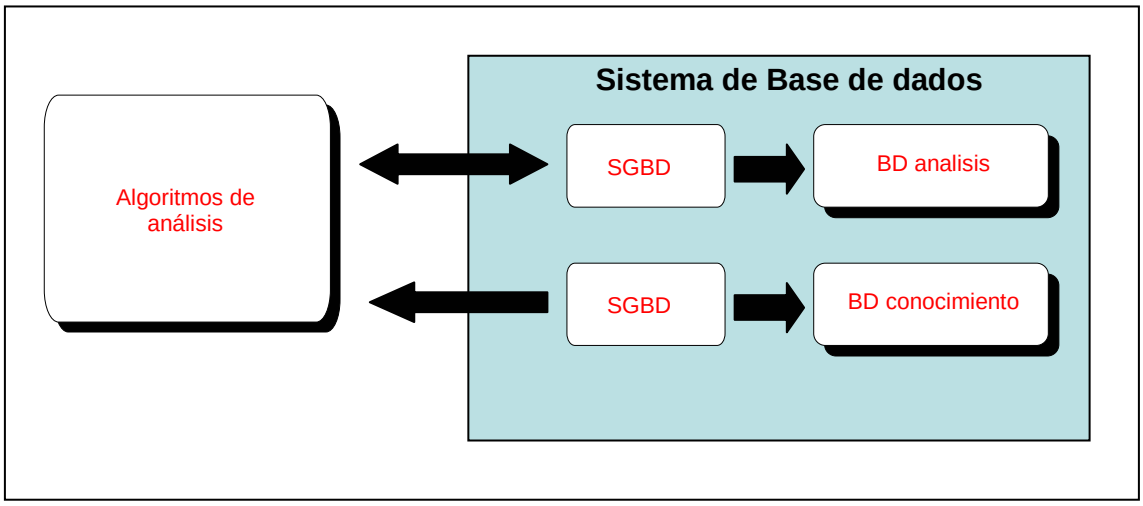


Figura B-0-8: Arquitectura del subsistema de bases de datos

A continuación se detalla brevemente la composición de cada uno de los subsistemas que componen las bases de datos de DiVA.

B.4.1 Almacenamiento de los datos de análisis

Este módulo, que en la implementación actual recibe el nombre de DataServer, se encarga de almacenar los datos, parciales o finales, arrojados por los algoritmos de análisis.

Además también se incluye información acerca de la configuración utilizada en la captura de las imágenes, procesado y etiquetado de estas. Dicha información puede contener diferentes parámetros según que dispositivo capturador utilicemos, que procesado o que etiquetado se realice.

Para implementar el DataServer se ha decidido desarrollar una base de datos en MySQL que permite almacenar los siguientes tipos de datos:

- Imágenes JPEG y sin comprimir
- Descripción del proceso de análisis y resultado del procesamiento de imagen (fichero xml)
- Máscaras binarias
- Parámetros de configuración de los FrameServers (fichero xml)

Para poder manejar la base de datos e interactuar con los elementos de la plataforma, se ha desarrollado un Sistema de Gestión de Bases de Datos (SGBD). Adicionalmente este SGBD realiza tareas de mantenimiento de la base de datos.

B.4.2 Almacenamiento del conocimiento

Este módulo, que en la implementación actual recibe el nombre de ContextServer, se encarga de proporcionar un contexto o dominio de aplicación a cada algoritmo. También incorpora un servidor IP para la comunicación con los algoritmos.

El modo de funcionamiento es similar al sistema de almacenamiento de datos de análisis. También se implementara como una base de datos pero tiene un propósito totalmente distinto al módulo anteriormente explicado. Solamente estará disponible para proporcionar información a los algoritmos antes de su etapa de procesado.

Esta base de datos también contara con un SGBD que actúa de interfaz con las distintas partes del sistema.

B.5 Arquitectura del sistema de presentación

La tarea de este subsistema es la de facilitar la presentación en un monitor de los datos generados por los algoritmos de análisis.

La implementación actual admite tres tipos de *displays* para mostrar los datos: uno basado en OpenCV que permite mostrar una cuadricula de imágenes, otro basado en tecnología MFC de Windows y los métodos del paquete HighGUI de OpenCV (que permite realizar otras operaciones como histogramas, asociar eventos con el ratón,…).

Si bien se dispone de tres métodos para presentar datos en pantalla, la plataforma *DiVA* no impone ninguna restricción sobre el método o manera de mostrar los resultados del procesado del módulo de análisis.

B.6 Arquitectura del sistema de procesado

El subsistema de procesado realiza todo el procesado de la plataforma DiVA. Dentro de este subsistema es posible integrar módulos de diferente naturaleza y con distintos objetivos.

El objetivo de este módulo es posibilitar la realización de un procesado distribuido en varios equipos o en un mismo equipo, incorporando la mínima sobrecarga al procesado de imagen.

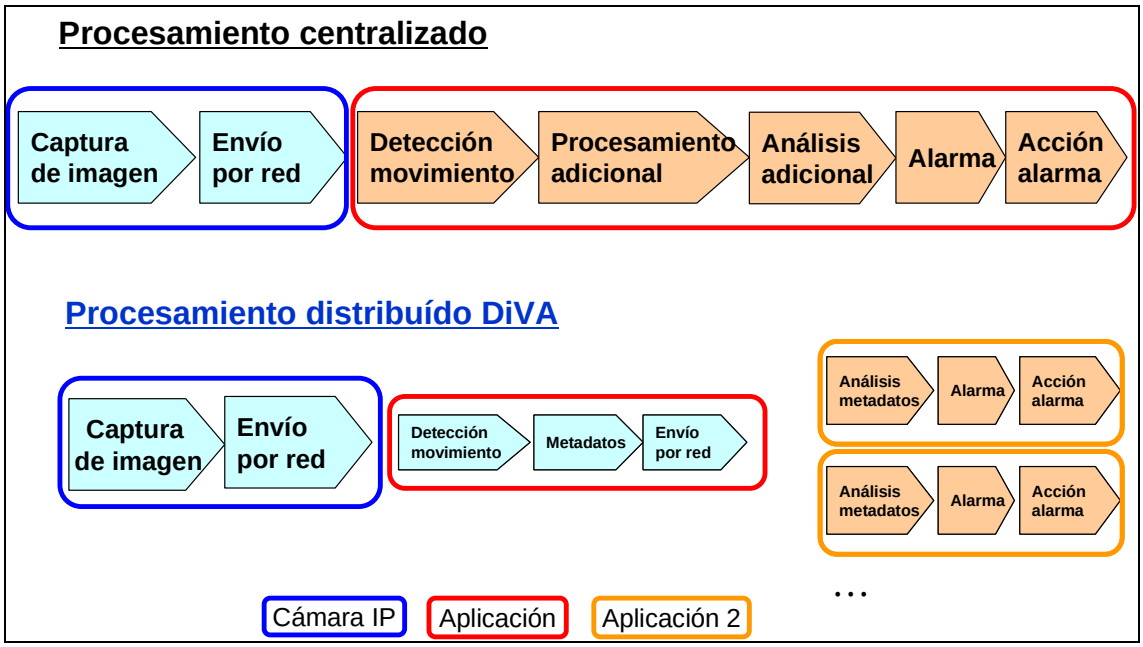


Figura B-0-9: Esquema de procesamiento distribuido DiVA

B.6.1 Encapsulador de proceso

Para gestionar el desarrollo de un nuevo componente de análisis de vídeo, se ha desarrollado un módulo que tiene la única función de encapsular los distintos algoritmos de análisis que pueden operar sobre la plataforma.

Este encapsulador ha sido desarrollado íntegramente en C/C++, siendo esta la única restricción impuesta por el diseño del sistema. El encapsulador tiene las siguientes funciones incorporadas:

1. Petición de datos:

    - Incorpora un cliente para conectarse al servidor de cuadros (FrameServer) seleccionado. Esta captura de datos puede ser secuencial o continua (en este caso se activa un buffer de cuadros)

    - Incorpora un cliente para conectarse al servidor de conocimiento y proporcionar un contexto de aplicación/modelos/… a la aplicación de análisis.

    - Incorpora un cliente para conectarse al servidor de información de análisis (datos multimedia) para poder acceder al procesado almacenado por otros algoritmos de análisis en esta base de datos.

    - Incorpora un método estándar de presentación de datos que permite mostrar varias imágenes en cuadrícula en una misma ventana.

- Incorpora un método para añadir el procesado de análisis de imagen. El procesado se puede añadir directamente dentro de la función o si ya disponemos de un algoritmo de análisis, realizar una llamada desde dicha función a este algoritmo.

En la siguiente figura se puede observar un esquema de los sistemas que están interconectados con el módulo de procesado.

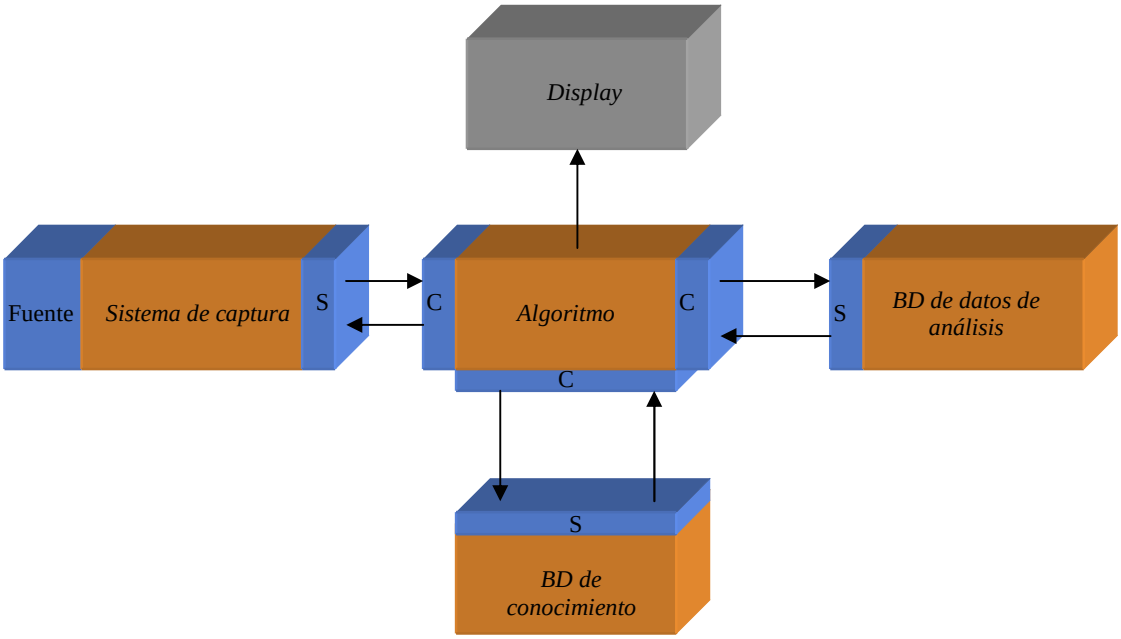


Figura B-0-10: Esquema de interconexión de un algoritmo en la plataforma DiVA

En la figura se puede observar como cada algoritmo desarrollado puede interactuar con tres tipos de servicios y con un módulo de presentación de resultados.

La intercomunicación entre los distintos algoritmos desarrollados y los servidores se realiza mediante el modelo cliente/servidor utilizando los clientes que incorpora el encapsulador.

Para cada algoritmo desarrollado y basándonos en el esquema (figura B-0-10) planteado anteriormente se deberán seguir los siguientes pasos:

- Petición de contexto de trabajo a la base de datos de conocimiento (ContextServer) (opcional)
    - Esta tarea inicial permite al algoritmo saber distintos detalles sobre el medio en el que trabaja o detalles sobre los objetos a detectar/reconocer.
- Petición de información (cuadros) al sistema de captura (FrameServer). (opcional)
    - Mediante dicha petición obtendremos la información que procesa nuestro algoritmo o parte de la información que necesita para realizar una determinada tarea. Esta información son los cuadros captados por las cámaras.
- Petición de información de análisis (cuadros analizados) a la base de datos de análisis (DataServer) (opcional)

- Algunos algoritmos no procesarán directamente la información proveniente del FrameServer, sino que la procesarán junto a otros datos procesados anteriormente (que se encuentran en el Data Server). La función del DataServer es la de proporcionar dichos datos de análisis anteriores (imágenes, descripciones,…).

- Ejecución del algoritmo
  - Esta parte no está definida en la arquitectura. Podrá realizarse de manera centralizada (en un solo PC) o distribuida (en varios PC's)

- Escritura de resultados en el Data Server
  - Dicho algoritmo guardará los resultados en el correspondiente servidor. Así pues dichos datos estarán disponibles para otros algoritmos que soliciten dichos datos. Adicionalmente también se guardara una referencia a la secuencia de datos original almacenada en el FrameServer

En el capítulo de desarrollo entraremos en detalle en el procedimiento para integrar/desarrollar nuevos componentes de procesado.

B.7 Requisitos de Software

Para realizar el desarrollo de la plataforma se han utilizado varias librerías externas. Estas librerías proporcionan diversas funcionalidades tales como manejo de conexiones TCP/IP, algoritmos de visión por computador...

Actualmente se utilizan los siguientes paquetes básicos:

- **Microsoft Fundation Classes (MFC)** 6.0. Estas librerías se usan para el manejo de sockets y el uso de hilos tanto en los clientes como en los servidores de la plataforma.

- **Intel® Open Source Computer Vision Library** versión 1.0 (OpenCV). El core de la plataforma DiVA esta desarrollado sobre el core de OpenCV. Se han modificado ciertas estructuras y métodos añadiendo nuevas funcionalidades.

Debido a las restricciones que imponen las librerías anteriormente mencionadas, la plataforma funciona bajo el sistema operativo Windows.

C Desarrollo de software para dispositivos PDA's

C.1. Desarrollo de aplicaciones en Windows Mobile

C.1.1. Introducción

Recordemos que en los objetivos de este PFC habíamos determinado que nuestro sistema inteligente de videoseguridad debía ser accesible a través del mayor número de plataformas posible. En cuanto a las plataformas móviles, habremos de escoger la que más extendida se encuentre en la actualidad y que más se adapte a nuestras necesidades.

Como ya habíamos visto en la sección 2.4.4 la mayoría del mercado de las PDAs eran dispositivos Pocket PC con S.O. de Windows. Estos dispositivos son precisamente los que nos interesan para el desarrollo de este PFC: ordenadores de bolsillo, con conectividad WLAN 802.11a/b/g (WiFi), capacidad de procesamiento alta (procesadores ARM con velocidades de entorno a los 400MHz) y con memoria ROM integrada. En estos dispositivos tenemos principalmente 3 opciones en cuanto a SS.OO.: Familiar Linux, Palm OS y Windows Mobile. Descartamos RIM OS por estar enfocado a dispositivos propios (Blackberry) sin conectividad WiFi y Symbian OS que, aunque soporta dispositivos de la familia ARM, estos suelen ser (en su mayoría) teléfonos móviles. No obstante, al menos por el momento, existe una pequeña variedad de Pocket PCs que soporten Familiar Linux con lo cual debemos descartarlo también.

De entre las dos posibilidades que nos quedan, nos quedaremos con Windows Mobile por dos razones. La primera es la compatibilidad de este S.O. con su homólogo en PC, Windows XP, para el cual desarrollaremos el sistema inteligente. La segunda razón es que Windows Mobile es un S.O. en constante actualización, lo que nos permitirá en el futuro mejorar nuestra aplicación de acuerdo con las mejoras y evoluciones de los SS.OO. de Microsoft en computadoras. En otras palabras, Windows Mobile siempre será compatible con el S.O. de Microsoft para PCs que exista en el momento.

C.1.2. La API de Windows (los SDKs)

En la URL http://msdn.microsoft.com/en-us/library/bb847935.aspx tenemos toda la información sobre el desarrollo de aplicaciones para Windows Mobile de la versión 2003 en adelante (hasta la 6.0). Desde esta primera versión, y con cada una de las siguientes, Microsoft provee a los programadores de sus SDKs para el desarrollo de aplicaciones tanto en código nativo (C++) como en otros códigos mantenidos (C#, Visual Basic .NET). Actualmente el último SDK de Windows es el de la versión 6, y tiene los siguientes requisitos para su uso:

- Windows XP SP2 / Vista.
- Microsoft Visual Studio 2005, Standard Edition superior.
- Microsoft .NET Compact Framework v2 SP1. SP2 recomendado.
- ActiveSync 4.5. ó Windows Mobile Device Center
- Windows Mobile 6 Professional SDK, o Windows Mobile 6 Standard SDK, o ambos.

Para el desarrollo de aplicaciones de terceros, los SDK de Windows incluyen múltiples ejemplos de código, además de todas las librerías y ficheros de cabecera necesarios. En lugar de numerar todas las posibilidades que nos ofrece el SDK (pues son demasiadas), enunciaremos las que tienen relación con la aplicación a desarrollar. Esto es, un cliente de streaming que permita visualizar (renderizar) video en tiempo real. Por tanto, nos centraremos en dos grandes áreas: conexiones y manejo de contenidos multimedia.

C.1.2.1 Conexiones remotas.

Para nuestro sistema de streaming tenemos dos opciones:

- Desarrollar una aplicación en bajo nivel que establezca una conexión extremo a extremo entre el servidor (PC) y el cliente (PDA).
- Utilizar sofware de nivel superior que nos permita abstraernos de las características del enlace de comunicaciones y podamos acceder directamente al contenido de la información transmitida por la red.

Para el primer caso, podemos desarrollar un programa que utilice la pila de protocolos del modelo TCP/IP, para el acceso al servidor tanto en redes locales como remotas. El SDK de Windows Mobile 5 (y 6) proporciona las siguientes interfaces:

- *Dynamic Host Configuration Protocol* (DHCP) *client*
- *Windows Internet Name Service* (WINS)
- *Domain Name System* (DNS) *client*
- *Dial-up* (PPP/SLIP) *suppor*t.
- TCP/IP *network printing*
- *Simple Network Management Protocol* (SNMP) *extension agent*
- *Wide area network* (WAN) *support*
- *Network Utilities*
- *Internet Protocol Helper* (IP Helper)
- *Windows* HTTP *Services* (WinHTTP)
- *Remote Procedure Call* (RPC)
- *Windows Internet* (WinInet)
- *Windows Sockets* (Winsock)

De estos nos centraremos en WinHTTP, WinInet y Winsock. El primer caso nos proporciona una interfaz de alto nivel para el protocolo de Internet HTTP/1.1.

WinHTTP puede ser utilizado para aplicaciones que se conectan a servidores HTTP, es decir, en aplicaciones basadas en clientes HTTP. WinHTTP es más seguro y robusto que WinInet, y debe ser utilizado para implementación de servidores y servicios (no es nuestro caso). WinInet proporciona funcionalidad basada en Internet incluyendo soporte para clientes FTP, HTTP, Passport, IE 6 y el compilador de MSXML. El API WinInet maneja toda la comunicación entre la aplicación y los Sockets de Windows (Winsock), abstrayendo al programador del mantenimiento y establecimiento de las sesiones de comunicaciones.

En el caso de implementar un cliente que conectara con un servidor de streaming HTTP, esto conllevaría desarrollar una aplicación que ejecutara en un navegador de Internet tipo Pocket Internet Explorer que incluyese un control ActiveX ó JavaScript para la visualización de la ventana de video.

Otra opción sería utilizar este mismo API (WinInet) con vistas a recibir los datos mediante un cliente FTP. Un servidor FTP instalado en la aplicación inteligente de videoseguridad enviaría bajo demanda las imágenes en cada instante, y la aplicación cliente en la PDA las mostraría por pantalla sin necesidad de controles ActiveX ó JavaScript.

La última opción evaluada es la de crear una aplicación de bajo nivel para establecer un enlace TCP/IP mediante Sockets de Windows. Utilizaríamos, pues el API Winsock del SDK de Windows Mobile. De este modo, todos los problemas derivados del establecimiento de las conexiones y sesiones de datos estarían ahora presentes. Esto aumentaría el tiempo de desarrollo significativamente, aunque cabe señalar que la mayoría de las primitivas son similares a las del API Win32, con lo que cabría intentar migrar alguna aplicación desarrollada para Windows (XP o Vista) de similares características. Además la ventaja principal de esta opción es que podemos tener exactamente la funcionalidad que deseemos en nuestra aplicación sin estar limitados por lo que nos ofrezcan las APIs de nivel superior.

C.1.2.2 Visualización de contenidos multimedia.

Volviendo a los objetivos del proyecto, recordamos que el usuario (que accede desde el dispositivo Windows Mobile) tiene que poder visualizar en tiempo real las imágenes que le envía la aplicación remota inteligente. Para ello tenemos dos opciones:

- Utilizar algún API de Windows para manejo de contenidos multimedia.
- Crear una aplicación web basada que contenga algún Objeto ActiveX, Windows Media, o similar para reproducir el stream de video.

Para el primer caso, el SDK de Windows Mobile 5 incluye las siguientes APIs:

- Direct3D Mobile (soporte para aplicaciones de gráficos 3D)
- DirectDraw (soporte para aplicaciones de gráficos 2D)
- DirectShow (soporte para reproducción y captura de contenidos multimedia)
- File-Based Digital Rights Management (contenidos con derechos de autor)
- Imaging for Windows Mobile-based Devices (soporte para varios códecs de imagenes estáticas).
- Speaker-Dependent Speech Support (para reconocimiento del habla).
- Waveform Audio for Windows Mobile-based Devices (control de entrada/salida de audio).
- Voice Recorder Control (grabación y reproducción de sonidos).
- Windows Mobile Games (soporte de programación de videojuegos).

Parece lógico que de entre estas APIs la que más se ajusta a nuestras necesidades es la de DirectShow, pues lo que queremos es reproducir un stream de video (preferentemente, video comprimido) enviado desde un servidor remoto.

DirectShow es una arquitectura "*middleware*" de tratamiento y reproducción de contenidos multimedia. Este framework multimedia permite facilitar la tarea de los programadores en diferentes aspectos:

- Procesado de streams que contienen grandes cantidades de datos a gran velocidad.
- Sincronización de audio, video y streams adicionales, para que se reproduzcan al mismo tiempo y a la misma velocidad.
- Sincronización de streams que provienen de distintas fuentes y orígenes, como archivos multimedia locales, redes de ordenadores, cámaras de video, difusión de televisión terrestre y otros dispositivos.
- Compresión / Descompresión de video en diferentes formatos (como AVI, MPEG, ASF, etc).

Esta arquitectura basada en el framework Microsoft Windows *Component Object Model* (COM), proporciona una interfaz para manejo de multimedia para distintos lenguajes de programación y está basado en filtros (DirectShow filters). La documentación y las herramientas para el desarrollo con DirectShow se encuentran en el SDK de Windows Mobile (a partir de 2003).

DirectShow divide un proceso de tratamiento de contenidos multimedia (por ejemplo, la reproducción), en una secuencia de etapas fundamentales representadas por filtros. Cada filtro tiene pines de entrada y/o salida para su conexión con otros filtros. La naturaleza de estas conexiones permite la implementación de múltiples funciones complejas de procesado.

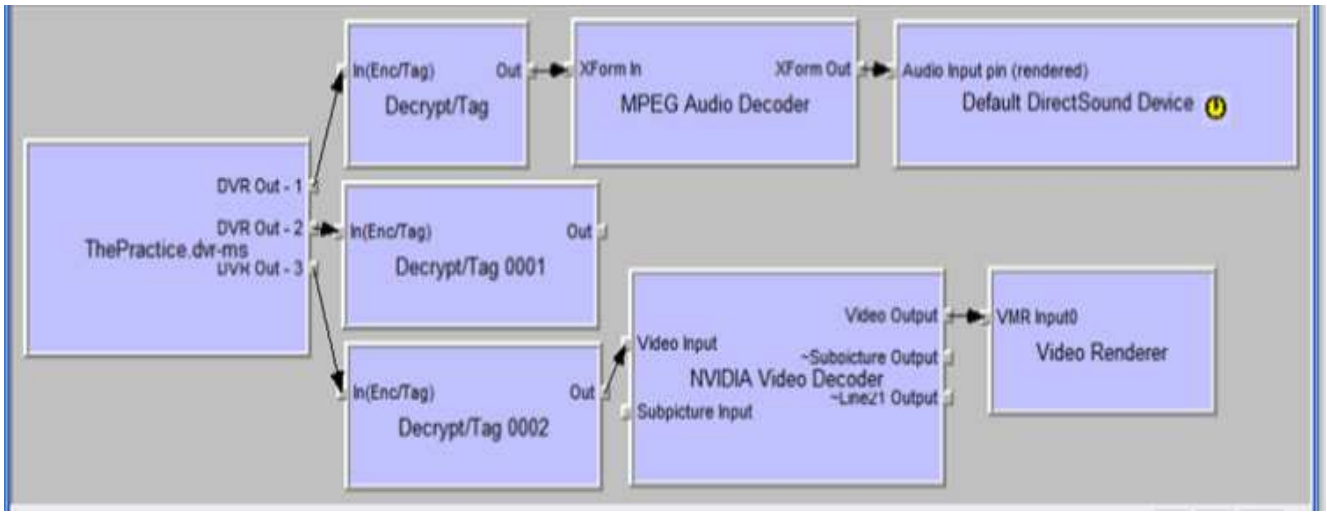


Figura C-0-11: Grafo de DirectShow elaborado con GraphEdit, herramienta de Dshow SDK para Windows XP. Representa la reproducción de video y audio procedente de una cámara de video digital.

Los filtros pueden ser implementados utilizando las DirectShow Base Classes, que son un conjunto de clases en C++ incluidas en el SDK.

Sin embargo, la cantidad de clases que Microsoft proporciona en el SDK de Windows Mobile es muy limitada, en comparación con el SDK de DirectShow (para versiones de Windows en PCs). Dicho de otro modo, para dispositivos móviles contamos con muy pocos filtros ya implementados, con lo cual las posibilidades son mucho menores. Si visitamos la web de la biblioteca de Microsoft (http://msdn.microsoft.com/en-us/library) en la sección de DirectShow para Windows Mobile podemos ver la lista de filtros incluidos. Entre ellos se incluyen:

- *File source filter* (async): actúan como fuente de datos a partir de archivos MOV, MPEG y WAV.
- Codecs de audio MPEG-1 y 2
- Soporte para MS RLE Video Codec y para descompresor AVI
- Video Splitters de MPEG y AVI: separan los streams de audio y video para su posterior procesado.
- WM ASF Writer Filter: filtro para volcado de contenidos multimedia con formatos Windows Media a ficheros.
- Video Renderering Filter: nos permite reproducir (renderizar) video no comprimido en pantalla.

Así que como vemos, las opciones que tenemos para nuestro desarrollo son pocas. Para empezar, necesitaríamos implementar un descompresor de video (por ejemplo MPEG-2 ó 4) pues el stream de video que vamos a recibir del servidor será de video comprimido. Necesitaríamos también un *splitter* que nos permita separar los diferentes streams que conforman el video. Ambas tareas complicarían enormemente nuestra tarea a la hora de crear la aplicación cliente para PDA.

La otra opción planteada para la creación del visor de video es la de una aplicación web multimedia con algún objeto (*applet*) incrustado tipo ActiveX o Windows Media. Analizaremos estas opciones en la siguiente subsección.

C.1.3. Applets para Windows Mobile

Un applet es un componente software que se ejecuta en el contexto de otro programa como por ejemplo, un navegador web. Normalmente el applet desempeña una función específica que a diferencia de un programa, no puede ejecutarse de manera independiente.

Existen varias empresas que ofrecen entornos de desarrollo de *applets* que desempeñan funciones de reproducción remota de contenidos multimedia. A continuación nombramos las más importantes:

- Adobe (antes Macromedia): Flash Lite 2.1 para Windows Mobile 5.0 CDK (*Content Development Kit*) y Macromedia Flash Player 7 para Windows Mobile CDK proporcionan técnicas y ejemplos de código para el desarrollo de contenido flash (multimedia, animaciones y gráficos) para Windows mobile, tanto en aplicaciones independientes a pantalla completa como embebidas en páginas web. Flash Player 7 for Pocket PC permite a los usuarios reproducir contenido Flash y ejecutar aplicaciones web multimedia Internet.
- Microsoft ActiveX: ActiveX es un *Component Object Model* desarrollado por Microsoft. Mediante estos COM, pueden crearse componentes software que realizan una determinada tarea o conjunto de tareas. Los controles ActiveX son pequeños programas que pueden usarse para crear aplicaciones distribuidas que puedan ser ejecutadas de manera remota a través de un navegador de Internet. Los controles ActiveX sólo son compatibles con Internet Explorer y en SS.OO. de Windows. Sin embargo no pueden ejecutarse en I.E Mobile ActiveX basados en Win32 (XP, Vista o anteriores), pues deben ser compilados para Windows Mobile con procesadores ARM. Actualmente la única opción para instalar controles ActiveX es descargarlos desde el PC e instarlos por medio de este en el dispositivo.
- Microsoft Windows Media Player: el SDK de Windows Media Player ofrece una gran cantidad de interfaces, aunque no todos están disponibles para Windows Mobile. De entre los sí disponibles se encuentra el reproductor embebido en Internet Explorer (útil para nuestra aplicación particular), que puede reproducir de forma video y audio de forma remota (con determinados formatos y contenedores, WMA y WMV principalmente).
- Sun Microsystems: Java Platform, Micro Edition (Java ME) proporciona un entorno flexible y robusto para aplicaciones que ejecutan en dispositivos móviles. Java ME SDK 3.0 es el software de desarrollo de Sun, que incluye el XML API for Java ME para creación de páginas web con contenido multimedia. Es un nuevo entorno de desarrollo basado en la plataforma Netbeans.

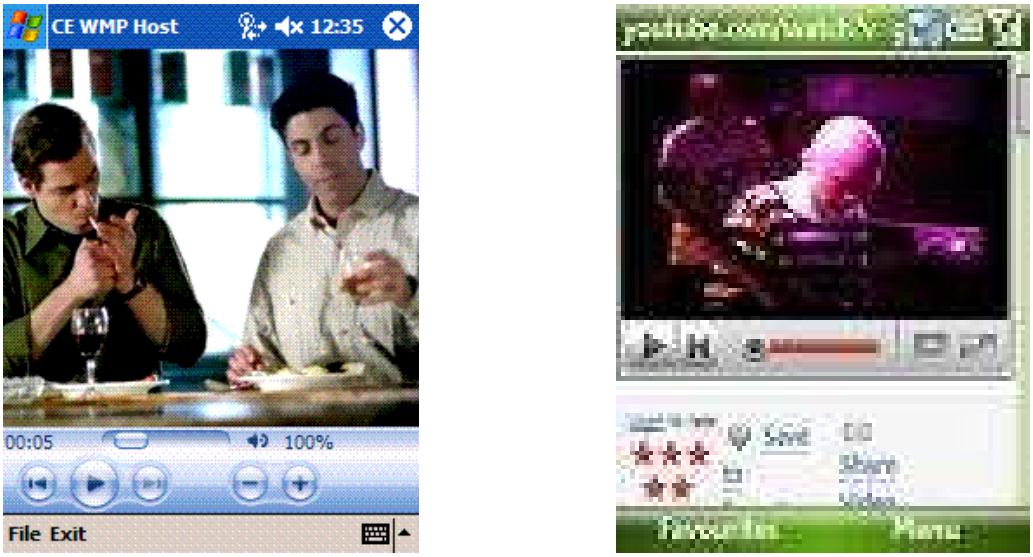


Figura C-0-12: Ejemplos de reproductores embebidos en Windows Mobile: Windows Media (izqda) y Adobe Flash (derecha)

C.1.4. los entornos de programación, etc.

Para el desarrollo de aplicaciones para Windows Mobile tenemos varias opciones sobretodo dependiendo del sistema operativo desde el cual pretendemos desarrollar el software. Aunque primero necesitamos algún mecanismo de sincronización o simplemente de comunicación entre el PC y la PDA para el envío de archivos e instalación remota de aplicaciones.

La principal herramienta de **Windows** para la sincronización con dispositivos Windows Mobile es ActiveSync, y más recientemente Windows Mobile Device Center. El proyecto SynCE permite conectar dispositivos Windows Mobile con ordenadores **Linux** (también *BSD y otros Unix) usando un cable USB o Bluetooth (aunque aún reconoce un número limitado de dispositivos del mercado). Desde **MACOS X** podemos acceder a la PDA con MissingSync, ó SyncMate sincronizando agenda, calendario, archivos de Office, fotos, etc.

En caso de utilizar sistemas MAC OS o GNU/Linux, es posible desarrollar software mediante entornos propios de cada uno que incluyan un compilador “cruzado” (cross-compiler) para el procesador específico del dispositivo (el más típico, *ARM-based*). Después debemos utilizar alguno de estos programas de sincronización con el dispositivo, que nos permitan enviar el ejecutable al dispositivo.

Como es previsible, en sistemas operativos Windows para PCs el desarrollo de software es mucho más sencillo pues la compatibilidad es absoluta y los entornos de programación disponen de varias herramientas de depuración que facilitan la corrección de errores. Existen dos entornos para desarrollo de software en Windows: Embedded visual c++ , y Microsoft Visual Studio.

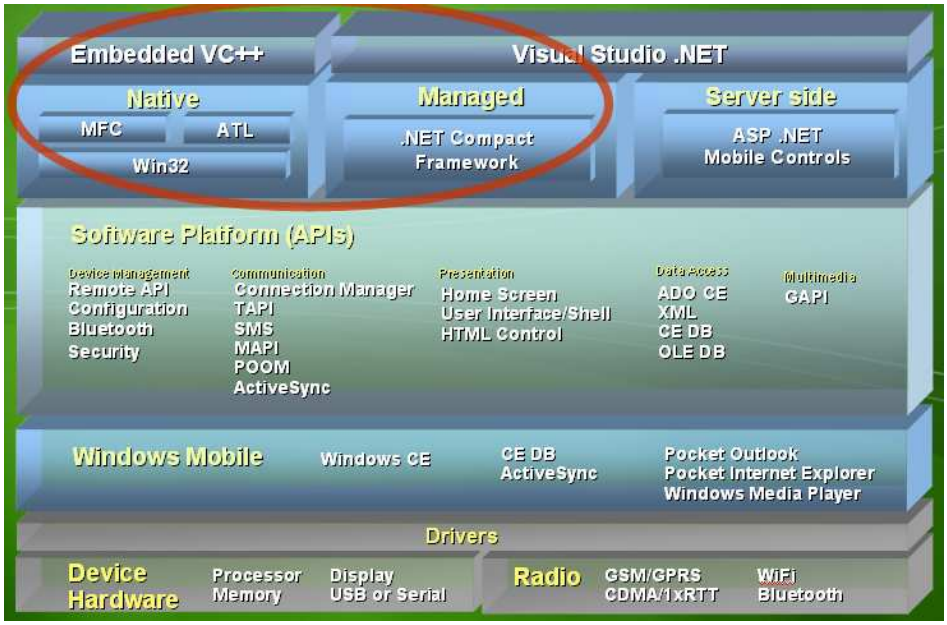


Figura C-0-13: plataforma actual de desarrollo en Windows Mobile (MSMDC 2004).

C.1.4.1. eMbedded Visual tools

eMbedded Visual Tools es un completo entorno de desarrollo para crear aplicaciones y componentes de sistema para dispositivos Windows Powered, incluyendo sistemas PocketPC y Smartphone (actualmente Windows Mobile).

La edición 2003 contiene Microsoft eMbedded Visual C++ 3.0, Microsoft eMbedded Visual Basic 3.0 y, también presenta los *kits* de desarrollo (SDKs) para PocketPC 2002 y Smartphone 2002. Actualmente existe la versión 4.0 del *pack*, junto con los SDKs de Windows Mobile 6.

La herramienta eMbedded Visual C++ 4.0 representa un entorno de desarrollo completo para creación de aplicaciones y componentes software en dispositivos con SS.OO basados en Windows CE. Incluye un emulador de Windows CE y Pocket PC 2003 para la ejecución y depuración de las aplicaciones desarrolladas. La última versión de este entorno existente es la 4.0 con service pack 4.

Esta herramienta fue utilizada para desarrollo para Windows CE y Pocket PC, y fue reemplazada por Microsoft Visual Studio 2005 y los SDKs de Windows Mobile.

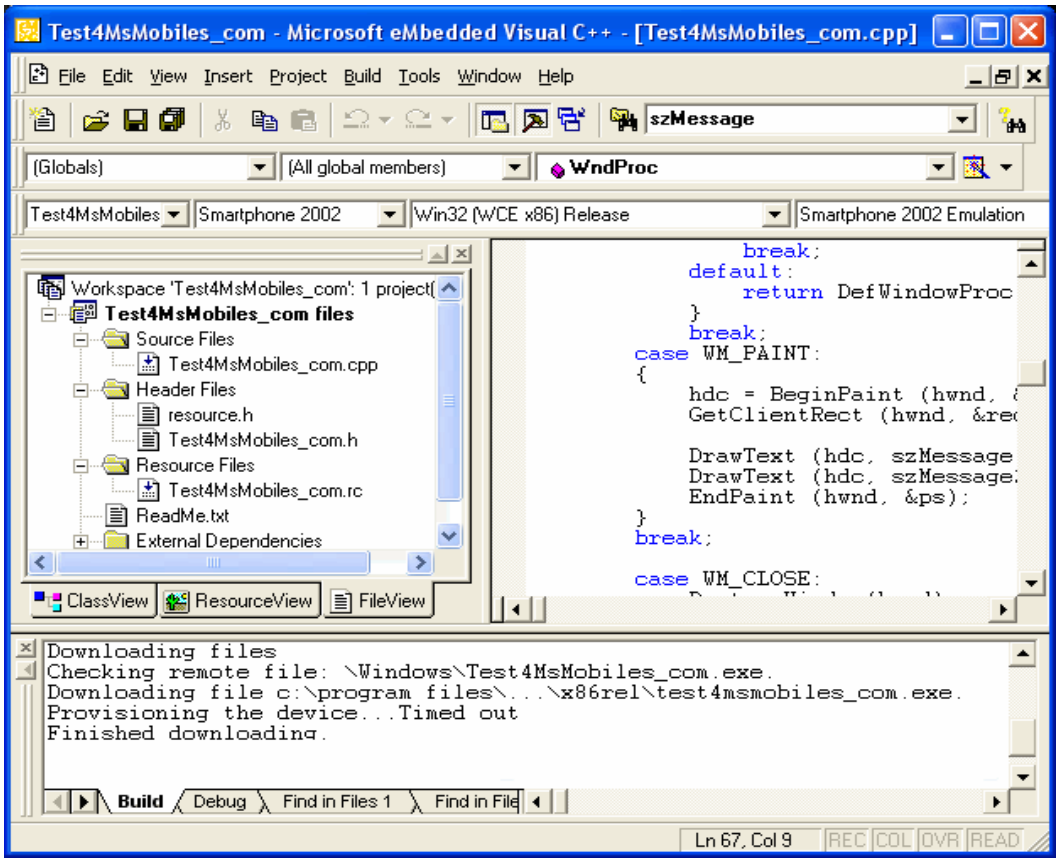


Figura C-0-14: entorno de desarrollo MS eVC++ 4.0, de apariencia muy similar a MS Visual C++.

C.1.4.2 Microsoft Visual Studio

Microsoft Visual Studio 2005 es el entorno líder en desarrollo para creación, prueba y utilización de aplicaciones para la plataforma Microsoft Windows. También proporciona el mismo nivel de soporte hacia la plataforma Windows Mobile.

Requiere la instalación de los kits de desarrollo (SDKs) correspondientes para cada familia de dispositivos Windows Mobile. Pueden instalarse múltiples SDKs en la misma computadora de desarrollo, correspondientes a las diferentes versiones de este S.O. Para desarrollo en WM 6.0 es posible utilizar el SDK de WM 5.0 y anteriores, sin embargo el SDK de WM 6.0 no debe usarse para desarrollo de aplicaciones para dispositivos con versiones de Windows Mobile antiguas. El SDK de WM 6 incluye nuevas funcionalidades y prestaciones que requieren dispositivos con WM 6 para poder funcionar.

Una de las características de Visual Studio para el desarrollo de aplicaciones para Windows Mobile es la opción *Deploy* que nos permite ejecutar la aplicación no solo en el emulador (figura C-0-16), característica compartida con embedded visual C++, sino en el

propio dispositivo de manera remota, junto con la posibilidad de depurar paso a paso el código de forma remota también.

Figura C-0-15: opciones de *deploy* (ejecución) en Visual Studio 2005

Figura C-0-16: emulador de un Smartphone con Windows Mobile 5

Los requisitos de este entorno son:

Windows Server 2003

- Windows Server 2003.
- Microsoft Visual Studio 2005, Standard Edition o superiores. SP1 recomendado.
- Microsoft .NET Compact Framework v2 SP1. SP2 recomendado.
- ActiveSync 4.5.
- Windows Mobile 6 Professional SDK, o Windows Mobile 6 Standard SDK, o ambos.

Windows XP

- Windows XP SP2.
- Microsoft Visual Studio 2005, Standard Edition o superior.
- Microsoft .NET Compact Framework v2 SP1. SP2 recomendado.
- ActiveSync 4.5.
- Windows Mobile 6 Professional SDK, o Windows Mobile 6 Standard SDK, o ambos.

Windows Vista

- Windows Vista.
- Microsoft Visual Studio 2005, Standard Edition o superior.
- Microsoft .NET Compact Framework v2 SP1. SP2 recomendado.
- Para la sincronización, se requiere Windows Mobile Device Center.
- Windows Mobile 6 Professional SDK, o Windows Mobile 6 Standard SDK, o ambos.

Como dijimos en la sección *C.1.2*., los SDKs de Windows Mobile proporcionan las librerías y APIs en los lenguajes C++, C# y Visual Basic .NET, y mediante el entorno Visual Studio 2005 podemos desarrollar en todos ellos sin la necesidad de herramientas y programas adicionales.

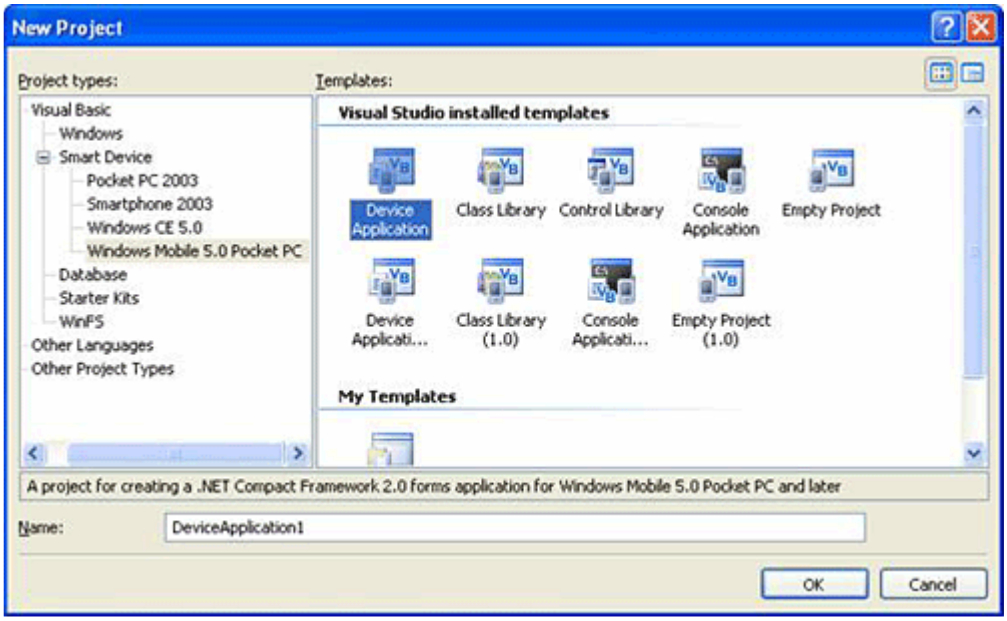


Figura C-0-17: Pantalla de selección de tipo de proyecto

C.2. *Reproductores / Clientes de Streaming comerciales*

C.2.1 Introducción

Aunque el reproductor más conocido y con mayor frecuencia de actualización para dispositivos Pocket PC sea Windows Media Placer, existen algunos otros, de código abierto o propietario, que desempeñan funcionalidades similares o incluso extienden las del reproductor de Microsoft. Haremos un breve resumen de ellos.

C.2.2 Pocket Windows Media Player

Windows Media Player Mobile ofrece:

- Reproducción de formatos de archivo *Windows Media Audio* (WMA), *Windows Media Video* (WMV), *Waveform Audio Format* (WAV) y MPEG-1 *Audio Layer* 3 (MP3)
- Reproducción tanto de archivos multimedia del dispositivo como de contenidos a través de la red introduciendo la URL del servidor de contenidos.
- Sincronización con la versión para PC de Windows Media Player (versiones 9 y superiores) para actualizar automáticamente sus listas de reproducción, archivos de música y vídeos favoritos en su dispositivo.

Windows Media Player Mobile está disponible para Windows Mobile 5.0 y Windows Mobile 6, y funciona en teléfonos con o sin pantalla táctil.

Figura C-0-18: captura de pantalla de WMP.

C.2.3 TCPMP

The Core Pocket Media Player (TCPMP) es un reproductor multimedia muy versátil con licencia dual cuyo desarrollo ha sido interrumpido (en este momento continua en desarrollo una versión comercial llamada CorePlayer y pronto aparecerá una de libre distribución de nombre BetaPlayer). Viene en una versión de código fuente cerrada y otra versión de código fuente abierta GPL. Soporta los siguientes sistemas operativos incluyendo Windows, Windows CE, Windows Mobile, Palm OS y Symbian OS. TCPMP soporta muchos decodificadores de audio y video, incluyendo AAC, ASF, ASX, AVI, DIVX, FLAC, H.263, H.264, M2V, Matroska, Monkey's Audio, MP2, MP3, Musepack, MPEG, MPG, Ogg, OGM, QuickTime, TTA, WAV, WavPack, Windows Media Video y Xvid. Estos pueden estar empotrados en navegadores Internet Explorer (para Windows, Windows Mobile y CE)).

Figura C-0-19: reproductor TCPMP

C.2.4 WinVibe

Reproductor multimedia para Windows Mobile que dispone de soporte para ficheros en formato MP3, Ogg Vorbis, WMA, ASF y WAV, además de poder hacer streaming por HTTP y Shoutcast. Otras de las funcionalidades que ofrece son gestor de listas de reproducción, efectos de sonido (3D, realce de bajos,...) y apagado automático.

La aplicación es gratuita y requiere, como mínimo, Windows Mobile 2003.

Figura C-0-20: Reproductor WinVibe en Windows Mobile 2003

C.2.5 VLC Mobile

Si visitamos la página web de VideoLAN aparentemente no existe la posibilidad de obtener este reproductor en versión Windows Mobile, puesto que ese proyecto ya no tiene soporte (la última compilación que podemos encontrar es de 2005). En la web http://wiki.videolan.org/ARM-XCompile se nos explica como compilar este reproductor para procesadores ARM-based (el código, no mantenido para dicha plataforma, puede descargarse de la URL de VideoLAN).

Aunque la documentación para este sistema si existe, y comenta que soporta casi todos los formatos multimedia que soportan las versiones Windows XP o GNU/Linux, dependiendo de cada versión podemos reproducir unos u otros formatos (mi experiencia personal es que sobretodo en temas de reproducción de streaming, los formatos aceptados dependen de la compilación del reproductor).

C.3. *Conclusiones*

En este anexo resumiremos los inconvenientes que nos han limitado en el desarrollo de este proyecto a la hora de desarrollar nuestra aplicación de videoseguridad, y que están reflejados en varias partes de esta memoria.

Puede observarse, al leer detenidamente este proyecto el peso que hemos dado a la documentación relativa a las PDAs. En un principio, el nombre del Proyecto Fin de Carrera no era el mismo que el actual. El titulo anterior era: "Adaptación de contenidos multimedia a terminales PDA". Fue necesario cambiar el título y los objetivos del Proyecto debido a limitaciones y problemas que resumiremos a continuación:

→ Limitaciones a la hora de desarrollar para el sistema operativo de Windows Mobile, como ya hemos visto en este Anexo C. Inicialmente se intentó adaptar una plataforma del grupo (VPULab) para la transmisión de "video IP a tasa variable" (VIPTV), pero no fue posible debido a las limitaciones del *framework* de *DirectShow* en Windows Mobile.

→ Limitaciones de dicho sistema operativo a la hora de reproducir video comprimido. Los reproductores disponibles actualmente para estos dispositivos (en particular para Windows Mobile) son pocos y admiten pocos formatos. Hubiera sido más apropiado utilizar un formato de video de poca tasa binaria (como por ejemplo H.264), que mejorara el rendimiento global del sistema.

→ Problemas a la hora de compilación de los programas. Aunque es posible desarrollar aplicaciones mediante eVC++, o VisualStudio (con el SDK de Windows Mobile), normalmente no es posible adaptar aplicaciones elaboradas en PC(Win32) por las incongruencias entre las funciones del API de WindowsMobile y Win32.

D Ficheros de configuración de CAIN

D.1 Fichero de entrada “raw_tcp_newitem_di.xml”

```
<?xml version="1.0" ?>
- <!--
 Example of DI that conveys a news item and its summary
  -->
- <DIDL xmlns="urn:mpeg:mpeg21:2002:02-DIDL-NS"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:cdi="urn:gti:cain-di" xmlns:mpeg7="urn:mpeg:mpeg7:schema:2004">
- <Item xsi:type="cdi:ItemType">
- <!--
 Original content
  -->
- <Component xsi:type="cdi:VideoComponentType" id="original">
- <Descriptor xsi:type="cdi:Mpeg7DescriptorType">
- <Statement mimeType="text/xml">
- <Mpeg7 xmlns="urn:mpeg:mpeg7:schema:2004">
- <Description xsi:type="MediaDescriptionType">
- <MediaInformation>
- <MediaProfile>
- <ComponentMediaProfile>
- <MediaFormat>
- <Content href="urn:mpeg:mpeg7:cs:ContentCS:2001:4">
<Name>Visual</Name>
</Content>
<BitRate>104857000</BitRate>
- <VisualCoding>
- <Format href="urn:mpeg:mpeg7:cs:VisualCodingFormatCS:2001:2.2.2"
  colorDomain="color">
<Name xml:lang="en">MPEG-2 Video Main Profile @ Main Level</Name>
</Format>
<Pixel aspectRatio="0.75" />
<Frame height="576" width="720" rate="25" />
</VisualCoding>
</MediaFormat>
</ComponentMediaProfile>
- <ComponentMediaProfile>
- <MediaFormat>
- <Content href="urn:mpeg:mpeg7:cs:ContentCS:2001:1">
<Name>Audio</Name>
</Content>
<BitRate>384000</BitRate>
- <AudioCoding>
- <Format href="urn:mpeg:mpeg7:cs:AudioCodingFormatCS:2001:4">
<Name xml:lang="en">MPEG-2 Audio Coding Format</Name>
</Format>
```

```
      </AudioCoding>
      </MediaFormat>
      </ComponentMediaProfile>
- <MediaFormat>
- <Content href="urn:mpeg:mpeg7:cs:ContentCS:2001:2">
  <Name>Audiovisual</Name>
      </Content>
- <FileFormat href="urn:mpeg:mpeg7:cs:FileFormatCS:2001:7">
  <Name>avi</Name>
      </FileFormat>
  <FileSize>45361152</FileSize>
  <BitRate>4843000</BitRate>
      </MediaFormat>
      </MediaProfile>
      </MediaInformation>
      </Description>
- <Description xsi:type="VariationDescriptionType">
- <VariationSet>
- <Source xsi:type="AudioVisualType">
- <AudioVisual>
- <MediaLocator>
  <MediaUri>#original</MediaUri>
      </MediaLocator>
      </AudioVisual>
      </Source>
- <Variation priority="1">
- <Content xsi:type="AudioVisualType">
- <AudioVisual>
- <MediaLocator>
  <MediaUri>#summarization</MediaUri>
      </MediaLocator>
      </AudioVisual>
      </Content>
  <VariationRelationship>summarization</VariationRelationship>
      </Variation>
      </VariationSet>
      </Description>
      </Mpeg7>
      </Statement>
      </Descriptor>
  <Resource mimeType="video/avi" ref="03.avi" />
      </Component>
  - <!--
    Summarized content
      -->
- <Component xsi:type="cdi:VideoComponentType" id="summarization">
- <Descriptor xsi:type="cdi:Mpeg7DescriptorType">
- <Statement mimeType="text/xml">
- <Mpeg7 xmlns="urn:mpeg:mpeg7:schema:2004">
- <Description xsi:type="MediaDescriptionType">
- <MediaInformation>
```

```
- <MediaProfile>
- <ComponentMediaProfile>
- <MediaFormat>
- <Content href="urn:mpeg:mpeg7:cs:ContentCS:2001:4">
  <Name>Visual</Name>
    </Content>
  <BitRate>120000</BitRate>
- <VisualCoding>
-          <Format          href="urn:uam:gti:cs:VisualCodingFormatCS:2008:raw"
    colorDomain="color">
  <Name xml:lang="en">Raw visual video</Name>
    </Format>
  <Pixel aspectRatio="0.75" />
  <Frame height="576" width="720" rate="25" />
    </VisualCoding>
    </MediaFormat>
    </ComponentMediaProfile>
- <ComponentMediaProfile>
- <MediaFormat>
- <Content href="urn:mpeg:mpeg7:cs:ContentCS:2001:1">
  <Name>Audio</Name>
    </Content>
  <BitRate>38400</BitRate>
- <AudioCoding>
- <Format href="urn:uam:gti:cs:AudioCodingFormatCS:2008:wav">
  <Name xml:lang="en">WAV audio</Name>
    </Format>
    </AudioCoding>
    </MediaFormat>
    </ComponentMediaProfile>
- <MediaFormat>
- <Content href="urn:mpeg:mpeg7:cs:ContentCS:2001:2">
  <Name>Audiovisual</Name>
    </Content>
- <FileFormat href="uam:gti:cs:FileformatCS:2008:summarized-video">
  <Name>GTI summarized video</Name>
    </FileFormat>
  <FileSize>93325312</FileSize>
  <BitRate>160443</BitRate>
    </MediaFormat>
    </MediaProfile>
    </MediaInformation>
    </Description>
    </Mpeg7>
    </Statement>
    </Descriptor>
                <Resource                mimeType="video/x-gti-summarized-video"
    ref="tcp://192.168.1.6:4090/mpeg2video.mpg" />
    </Component>
    </Item>
    </DIDL>
```

En el lugar resaltado en amarillo debemos indicar la IP y puerto del DiVAServer incluido en el proyecto DiVAMultiSurveillanceSystem (el puerto por defecto es el 4090).

D.2 Fichero de configuración de terminales “mesh-ued.xml”

```
<?xml version="1.0" ?>
- <!--
 Terminal description according to MESH requeriments
     See S4.1.1 Table 1 for details

 -->
- <DIA xmlns="urn:mpeg:mpeg21:2003:01-DIA-NS"
 xmlns:mpeg7="urn:mpeg:mpeg7:schema:2001"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xmlns:cde="urn:gti:cain-description-elements">
- <Description xsi:type="UsageEnvironmentType">
- <UsageEnvironmentProperty xsi:type="cde:TerminalsType">
- <Terminal id="audiovisual_mobile_1" xsi:type="cde:TerminalType">
- <TerminalCapability xsi:type="cde:HandlerCapabilitiesType">
<Handler handlerURI="urn:mpeg:mpeg21:2007:01-BBL-NS:handler:FILE" />
</TerminalCapability>
- <TerminalCapability xsi:type="cde:CodecCapabilitiesType">
- <Decoding xsi:type="VideoCapabilitiesType">
- <Format href="urn:mpeg:mpeg7:cs:VisualCodingFormatCS:2001:3.1.2">
<mpeg7:Name xml:lang="en">MPEG-4 Visual Simple Profile @ Level 1</mpeg7:Name>
</Format>
- <CodecParameter xsi:type="CodecParameterBitRateType">
<BitRate>32000</BitRate>
</CodecParameter>
</Decoding>
- <Decoding xsi:type="AudioCapabilitiesType">
- <Format href="urn:mpeg:mpeg7:cs:AudioCodingFormatCS:2001:6">
<mpeg7:Name xml:lang="en">AMR</mpeg7:Name>
</Format>
- <CodecParameter xsi:type="CodecParameterBitRateType">
<BitRate>7950</BitRate>
</CodecParameter>
</Decoding>
</TerminalCapability>
- <TerminalCapability xsi:type="cde:DisplaysType">
- <Display>
- <DisplayCapability xsi:type="cde:DisplayCapabilityType" colorCapable="true">
- <Mode refreshRate="15">
<Resolution horizontal="176" vertical="144" />
</Mode>
</DisplayCapability>
```

```
    </Display>
    </TerminalCapability>
    </Terminal>
- <Terminal id="audiovisual_mobile_2" xsi:type="cde:TerminalType">
- <TerminalCapability xsi:type="cde:HandlerCapabilitiesType">
  <Handler handlerURI="urn:mpeg:mpeg21:2007:01-BBL-NS:handler:FILE" />
    </TerminalCapability>
- <TerminalCapability xsi:type="cde:CodecCapabilitiesType">
- <Decoding xsi:type="VideoCapabilitiesType">
- <Format href="urn:gti:cs:VisualCodingFormatCS:2007:1">
  <mpeg7:Name xml:lang="en">H.264 Baseline Profile @ Level 1.1</mpeg7:Name>
    </Format>
- <CodecParameter xsi:type="CodecParameterBitRateType">
  <BitRate>192000</BitRate>
    </CodecParameter>
    </Decoding>
- <Decoding xsi:type="AudioCapabilitiesType">
- <Format href="urn:mpeg:mpeg7:cs:AudioCodingFormatCS:2001:4.3.1">
     <mpeg7:Name xml:lang="en">MPEG-2 Audio AAC Low Complexity
    Profile</mpeg7:Name>
    </Format>
- <CodecParameter xsi:type="CodecParameterBitRateType">
  <BitRate>32000</BitRate>
    </CodecParameter>
    </Decoding>
    </TerminalCapability>
- <TerminalCapability xsi:type="cde:DisplaysType">
- <Display>
- <DisplayCapability xsi:type="cde:DisplayCapabilityType" colorCapable="true">
- <Mode refreshRate="15">
  <Resolution horizontal="352" vertical="288" />
    </Mode>
    </DisplayCapability>
    </Display>
    </TerminalCapability>
    </Terminal>
- <Terminal id="audiovisual_desktop_1">
- <TerminalCapability xsi:type="cde:HandlerCapabilitiesType">
  <Handler handlerURI="urn:mpeg:mpeg21:2007:01-BBL-NS:handler:FILE" />
    </TerminalCapability>
- <TerminalCapability xsi:type="cde:CodecCapabilitiesType">
- <Decoding xsi:type="VideoCapabilitiesType">
- <Format href="urn:mpeg:mpeg7:cs:VisualCodingFormatCS:2001:3.1.2">
     <mpeg7:Name xml:lang="en">MPEG-4 Visual Simple Profile @ Level
    1</mpeg7:Name>
    </Format>
- <CodecParameter xsi:type="CodecParameterBitRateType">
  <BitRate>320000</BitRate>
    </CodecParameter>
    </Decoding>
- <Decoding xsi:type="AudioCapabilitiesType">
- <Format href="urn:mpeg:mpeg7:cs:AudioCodingFormatCS:2001:4.3">
  <mpeg7:Name xml:lang="en">MPEG-2 Audio AAC</mpeg7:Name>
```

```
    </Format>
- <CodecParameter xsi:type="CodecParameterBitRateType">
  <BitRate>64000</BitRate>
    </CodecParameter>
    </Decoding>
    </TerminalCapability>
- <TerminalCapability xsi:type="cde:DisplaysType">
- <Display>
- <DisplayCapability xsi:type="cde:DisplayCapabilityType" colorCapable="true">
- <Mode refreshRate="15">
  <Resolution horizontal="352" vertical="288" />
    </Mode>
    </DisplayCapability>
    </Display>
    </TerminalCapability>
    </Terminal>
- <Terminal id="audiovisual_desktop_2">
- <TerminalCapability xsi:type="cde:HandlerCapabilitiesType">
  <Handler handlerURI="urn:mpeg:mpeg21:2007:01-BBL-NS:handler:FILE" />
    </TerminalCapability>
- <TerminalCapability xsi:type="cde:CodecCapabilitiesType">
- <Decoding xsi:type="VideoCapabilitiesType">
- <Format href="urn:gti:cs:VisualCodingFormatCS:2007:2">
  <mpeg7:Name xml:lang="en">H.264 Video Main Profile @ Level 3</mpeg7:Name>
    </Format>
- <CodecParameter xsi:type="CodecParameterBitRateType">
  <BitRate>768000</BitRate>
    </CodecParameter>
    </Decoding>
- <Decoding xsi:type="AudioCapabilitiesType">
- <Format href="urn:mpeg:mpeg7:cs:AudioCodingFormatCS:2001:4.3">
  <mpeg7:Name xml:lang="en">MPEG-2 Audio AAC</mpeg7:Name>
    </Format>
- <CodecParameter xsi:type="CodecParameterBitRateType">
  <BitRate>96000</BitRate>
    </CodecParameter>
    </Decoding>
    </TerminalCapability>
- <TerminalCapability xsi:type="cde:DisplaysType">
- <Display>
- <DisplayCapability xsi:type="cde:DisplayCapabilityType" colorCapable="true">
- <Mode refreshRate="25">
  <Resolution horizontal="720" vertical="576" />
    </Mode>
    </DisplayCapability>
    </Display>
    </TerminalCapability>
    </Terminal>
- <Terminal id="online_web">
- <TerminalCapability xsi:type="cde:HandlerCapabilitiesType">
  <Handler handlerURI="urn:mpeg:mpeg21:2007:01-BBL-NS:handler:HTTP" />
    </TerminalCapability>
- <TerminalCapability xsi:type="cde:CodecCapabilitiesType">
```

```
- <Decoding xsi:type="VideoCapabilitiesType">
- <Format href="urn:mpeg:mpeg7:cs:VisualCodingFormatCS:2001:3.1.2">
    <mpeg7:Name xml:lang="en">MPEG-4 Visual Simple Profile @ Level 1</mpeg7:Name>
    </Format>
- <Format href="urn:gti:cs:VisualCodingFormatCS:2007:2">
  <mpeg7:Name xml:lang="en">H.264 Video Main Profile @ Level 3</mpeg7:Name>
    </Format>
- <CodecParameter xsi:type="CodecParameterBitRateType">
  <BitRate>768000</BitRate>
    </CodecParameter>
    </Decoding>
- <Decoding xsi:type="AudioCapabilitiesType">
- <Format href="urn:mpeg:mpeg7:cs:AudioCodingFormatCS:2001:4.3.1">
    <mpeg7:Name xml:lang="en">MPEG-2 Audio AAC Low Complexity Profile</mpeg7:Name>
    </Format>
- <CodecParameter xsi:type="CodecParameterBitRateType">
  <BitRate>96000</BitRate>
    </CodecParameter>
    </Decoding>
    </TerminalCapability>
- <TerminalCapability xsi:type="cde:DisplaysType">
- <Display>
- <DisplayCapability xsi:type="cde:DisplayCapabilityType" colorCapable="true">
- <Mode refreshRate="25">
  <Resolution horizontal="352" vertical="288" />
  <Resolution horizontal="640" vertical="480" />
    </Mode>
    </DisplayCapability>
    </Display>
    </TerminalCapability>
    </Terminal>
- <Terminal id="mpeg2_adapted_online_web">
- <TerminalCapability xsi:type="cde:HandlerCapabilitiesType">
  <Handler handlerURI="urn:mpeg:mpeg21:2007:01-BBL-NS:handler:HTTP" />
    </TerminalCapability>
- <TerminalCapability xsi:type="cde:CodecCapabilitiesType">
- <Decoding xsi:type="VideoCapabilitiesType">
- <Format href="urn:mpeg:mpeg7:cs:VisualCodingFormatCS:2001:2.2.2">
    <mpeg7:Name xml:lang="en">MPEG-2 Video Main Profile @ Main Level</mpeg7:Name>
    </Format>
  <CodecParameter xsi:type="CodecParameterBitRateType" />
    </Decoding>
- <Decoding xsi:type="AudioCapabilitiesType">
- <Format href="urn:mpeg:mpeg7:cs:AudioCodingFormatCS:2001:4">
  <mpeg7:Name xml:lang="en">>MPEG-2 Audio Coding Format</mpeg7:Name>
    </Format>
  <CodecParameter xsi:type="CodecParameterBitRateType" />
    </Decoding>
    </TerminalCapability>
- <TerminalCapability xsi:type="cde:DisplaysType">
```

```
- <Display>
- <DisplayCapability xsi:type="cde:DisplayCapabilityType" colorCapable="true">
- <Mode refreshRate="25">
  <Resolution horizontal="720" vertical="576" />
    </Mode>
    </DisplayCapability>
    </Display>
    </TerminalCapability>
    </Terminal>
- <Terminal id="h264_adapted_online_web">
- <TerminalCapability xsi:type="cde:HandlerCapabilitiesType">
  <Handler handlerURI="urn:mpeg:mpeg21:2007:01-BBL-NS:handler:HTTP" />
    </TerminalCapability>
- <TerminalCapability xsi:type="cde:CodecCapabilitiesType">
- <Decoding xsi:type="VideoCapabilitiesType">
- <Format href="urn:gti:cs:VisualCodingFormatCS:2007:1">
  <mpeg7:Name xml:lang="en">H.264 Baseline Profile @ Level 1.1</mpeg7:Name>
    </Format>
- <CodecParameter xsi:type="CodecParameterBitRateType">
  <BitRate>192000</BitRate>
    </CodecParameter>
    </Decoding>
- <Decoding xsi:type="AudioCapabilitiesType">
- <Format href="urn:mpeg:mpeg7:cs:AudioCodingFormatCS:2001:4.3.1">
     <mpeg7:Name xml:lang="en">MPEG-2 Audio AAC Low Complexity
    Profile</mpeg7:Name>
    </Format>
- <CodecParameter xsi:type="CodecParameterBitRateType">
  <BitRate>32000</BitRate>
    </CodecParameter>
    </Decoding>
    </TerminalCapability>
- <TerminalCapability xsi:type="cde:DisplaysType">
- <Display>
- <DisplayCapability xsi:type="cde:DisplayCapabilityType" colorCapable="true">
- <Mode refreshRate="15">
  <Resolution horizontal="352" vertical="288" />
    </Mode>
    </DisplayCapability>
    </Display>
    </TerminalCapability>
    </Terminal>
    </UsageEnvironmentProperty>
  - <!--
    Networks description
    -->
- <UsageEnvironmentProperty xsi:type="cde:NetworksType">
- <Network id="modem" xsi:type="cde:NetworkType">
          <NetworkCharacteristic xsi:type="cde:NetworkCapabilityType"
    maxCapacity="56000" />
    </Network>
- <Network id="adsl" xsi:type="cde:NetworkType">
```

```
            <NetworkCharacteristic xsi:type="cde:NetworkCapabilityType"
    maxCapacity="1024000" minGuaranteed="96000" />
    </Network>
    </UsageEnvironmentProperty>
  - <!--
    Users description
    -->
- <UsageEnvironmentProperty xsi:type="cde:UsersType">
- <User id="reporter" xsi:type="UserType">
- <UserCharacteristic xsi:type="UsagePreferencesType">
- <UsagePreferences>
-                                          <mpeg7:FilteringAndSearchPreferences
    xsi:type="cde:FilteringAndSearchPreferencesType">
- <cde:SourcePreferences>
- <cde:MediaFormat preferenceValue="50">
- <mpeg7:Content href="urn:mpeg:mpeg7:cs:ContentCS:2001:2">
  <mpeg7:Name>Audiovisual</mpeg7:Name>
    </mpeg7:Content>
-         <cde:FileFormat href="urn:mpeg:mpeg7:cs:FileFormatCS:2001:5"
    preferenceValue="50">
  <cde:Name>MPEG-4 file format</cde:Name>
    </cde:FileFormat>
-         <cde:FileFormat href="urn:mpeg:mpeg7:cs:FileFormatCS:2001:3"
    preferenceValue="0">
  <cde:Name>MPEG file format</cde:Name>
    </cde:FileFormat>
-         <cde:FileFormat href="urn:mpeg:mpeg7:cs:FileFormatCS:2001:7"
    preferenceValue="-50">
  <cde:Name>Audio video interleave format</cde:Name>
    </cde:FileFormat>
- <cde:VisualCoding preferenceValue="30">
-     <cde:Format href="urn:mpeg:mpeg7:cs:VisualCodingFormatCS:2001:3.1"
    preferenceValue="50">
  <cde:Name>MPEG-4 Visual Simple Profile</cde:Name>
    </cde:Format>
-     <cde:Format href="urn:mpeg:mpeg7:cs:VisualCodingFormatCS:2001:2.2.2"
    preferenceValue="0">
  <cde:Name>MPEG-2 Video Main Profile @ Main Level</cde:Name>
    </cde:Format>
    </cde:VisualCoding>
- <cde:AudioCoding preferenceValue="30">
-       <cde:Format href="urn:mpeg:mpeg7:cs:AudioCodingFormatCS:2001:6"
    preferenceValue="50">
  <cde:Name>AMR</cde:Name>
    </cde:Format>
-     <cde:Format href="urn:mpeg:mpeg7:cs:AudioCodingFormatCS:2001:4.3"
    preferenceValue="0">
  <cde:Name>MPEG-2 Audio AAC</cde:Name>
    </cde:Format>
    </cde:AudioCoding>
    </cde:MediaFormat>
    </cde:SourcePreferences>
    </mpeg7:FilteringAndSearchPreferences>
```

```
    </UsagePreferences>
    </UserCharacteristic>
    </User>
- <User id="cain" xsi:type="UserType">
- <UserCharacteristic xsi:type="UsagePreferencesType">
- <UsagePreferences>
-                                                    <mpeg7:FilteringAndSearchPreferences
    xsi:type="cde:FilteringAndSearchPreferencesType">
- <cde:SourcePreferences>
- <cde:MediaFormat preferenceValue="50">
- <mpeg7:Content href="urn:mpeg:mpeg7:cs:ContentCS:2001:2">
  <mpeg7:Name>Audiovisual</mpeg7:Name>
    </mpeg7:Content>
-          <cde:FileFormat          href="urn:mpeg:mpeg7:cs:FileFormatCS:2001:3"
    preferenceValue="50">
  <cde:Name>MPEG file format</cde:Name>
    </cde:FileFormat>
-          <cde:FileFormat          href="urn:mpeg:mpeg7:cs:FileFormatCS:2001:7"
    preferenceValue="0">
  <cde:Name>Audio video interleave format</cde:Name>
    </cde:FileFormat>
-          <cde:FileFormat          href="urn:mpeg:mpeg7:cs:FileFormatCS:2001:5"
    preferenceValue="-50">
  <cde:Name>MPEG-4 file format</cde:Name>
    </cde:FileFormat>
- <cde:VisualCoding preferenceValue="30">
-    <cde:Format      href="urn:mpeg:mpeg7:cs:VisualCodingFormatCS:2001:2.2.2"
    preferenceValue="50">
  <cde:Name>MPEG-2 Video Main Profile @ Main Level</cde:Name>
    </cde:Format>
-      <cde:Format      href="urn:mpeg:mpeg7:cs:VisualCodingFormatCS:2001:3.1"
    preferenceValue="0">
  <cde:Name>MPEG-4 Visual Simple Profile</cde:Name>
    </cde:Format>
-      <cde:Format      href="urn:mpeg:mpeg7:cs:VisualCodingFormatCS:2001:3.2"
    preferenceValue="-10">
  <cde:Name>MPEG-4 Visual Advanced Simple Profile</cde:Name>
    </cde:Format>
    </cde:VisualCoding>
- <cde:AudioCoding preferenceValue="30">
-      <cde:Format      href="urn:mpeg:mpeg7:cs:AudioCodingFormatCS:2001:4.3"
    preferenceValue="50">
  <cde:Name>MPEG-2 Audio AAC</cde:Name>
    </cde:Format>
-       <cde:Format       href="urn:mpeg:mpeg7:cs:AudioCodingFormatCS:2001:6"
    preferenceValue="0">
  <cde:Name>AMR</cde:Name>
    </cde:Format>
    </cde:AudioCoding>
    </cde:MediaFormat>
    </cde:SourcePreferences>
- <cde:AdaptationPreferences>
  <cde:ContentDegradation preferenceValue="80" />
```

```
<cde:Online preferenceValue="50" />
<cde:MinConversions preferenceValue="30" />
  </cde:AdaptationPreferences>
  </mpeg7:FilteringAndSearchPreferences>
  </UsagePreferences>
  </UserCharacteristic>
  </User>
  </UsageEnvironmentProperty>
  </Description>
  </DIA>
```

Señalado en amarillo el lugar donde empieza la descripción del terminal destino que utiliza el programa RawVideoCombinerTestCase

D.3 Fichero de salida “ovdi.xml”

```
<?xml version="1.0" ?>
- <DIDL xmlns="urn:mpeg:mpeg21:2002:02-DIDL-NS"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:cdi="urn:gti:cain-di" xmlns:mpeg7="urn:mpeg:mpeg7:schema:2004">
- <Item xsi:type="cdi:ItemType">
- <Component xsi:type="cdi:VideoComponentType" id="id1">
  - <!--
   Component Descriptors
   -->
- <Descriptor xsi:type="cdi:Mpeg7DescriptorType">
- <Statement mimeType="text/xml">
- <Mpeg7 xmlns="urn:mpeg:mpeg7:schema:2004">
- <Description xsi:type="MediaDescriptionType">
- <MediaInformation>
- <MediaProfile>
- <ComponentMediaProfile>
- <MediaFormat>
- <Content href="urn:mpeg:mpeg7:cs:ContentCS:2001:4">
  <Name>Visual</Name>
  </Content>
  <BitRate>5000</BitRate>
- <VisualCoding>
  <Format href="urn:mpeg:mpeg7:cs:VisualCodingFormatCS:2001:2.2.2"
  colorDomain="color" />
  <Pixel aspectRatio="0.75" />
  <Frame height="576" width="720" rate="25" />
  </VisualCoding>
  </MediaFormat>
  </ComponentMediaProfile>
- <ComponentMediaProfile>
- <MediaFormat>
```

```
- <Content href="urn:mpeg:mpeg7:cs:ContentCS:2001:1">
  <Name>Audio</Name>
    </Content>
  <BitRate>2000</BitRate>
- <AudioCoding>
  <Format href="urn:mpeg:mpeg7:cs:AudioCodingFormatCS:2001:4" />
    </AudioCoding>
    </MediaFormat>
    </ComponentMediaProfile>
- <MediaFormat>
- <Content href="urn:mpeg:mpeg7:cs:ContentCS:2001:2">
  <Name>Audioviual</Name>
    </Content>
- <FileFormat href="urn:mpeg:mpeg7:cs:FileFormatCS:2001:3">
  <Name>video/mpeg</Name>
    </FileFormat>
  <FileSize>93325312</FileSize>
  <BitRate>20000</BitRate>
    </MediaFormat>
    </MediaProfile>
    </MediaInformation>
    </Description>
    </Mpeg7>
    </Statement>
    </Descriptor>
  - <!--
    The resource itself
    -->
   <Resource mimeType="video/mpeg" ref="http://192.168.1.7:8085/out_r1.mpg"
    />
    </Component>
    </Item>
    </DIDL>
```

Señalado en amarillo el lugar donde aparece la URL y puerto de servicio del Servidor de Video Procesado.

PRESUPUESTO

1) Ejecución Material

- Compra de ordenador personal (Software incluido)..... ..... 2.000 €
- Alquiler de impresora láser durante 6 meses ..... 50 €
- Material de oficina ..... 150 €
- Total de ejecución material ..... 2.200 €

2) Gastos generales

- 16 % sobre Ejecución Material ..... 352 €

3) Beneficio Industrial

- 6 % sobre Ejecución Material ..... 132 €

4) Honorarios Proyecto

- 640 horas a 15 € / hora ..... 9600 €

5) Material fungible

- Gastos de impresión ..... 60 €
- Encuadernación ..... 200 €

6) Subtotal del presupuesto

- Subtotal Presupuesto ..... 12060 €

7) I.V.A. aplicable

- 16% Subtotal Presupuesto ..... 1929.6 €

8) Total presupuesto

- Total Presupuesto ..... 13989,6 €

Madrid, Febrero de 2009

El Ingeniero Jefe de Proyecto

Fdo.: Guillermo López-Oliva Santa Cruz
Ingeniero Superior de Telecomunicación

PLIEGO DE CONDICIONES

Este documento contiene las condiciones legales que guiarán la realización, en este proyecto, de un Sistema Inteligente de Gestión de Múltiples Cámaras de Videoseguridad. En lo que sigue, se supondrá que el proyecto ha sido encargado por una empresa cliente a una empresa consultora con la finalidad de realizar dicho sistema. Dicha empresa ha debido desarrollar una línea de investigación con objeto de elaborar el proyecto. Esta línea de investigación, junto con el posterior desarrollo de los programas está amparada por las condiciones particulares del siguiente pliego.

Supuesto que la utilización industrial de los métodos recogidos en el presente proyecto ha sido decidida por parte de la empresa cliente o de otras, la obra a realizar se regulará por las siguientes:

Condiciones generales

1. La modalidad de contratación será el concurso. La adjudicación se hará, por tanto, a la proposición más favorable sin atender exclusivamente al valor económico, dependiendo de las mayores garantías ofrecidas. La empresa que somete el proyecto a concurso se reserva el derecho a declararlo desierto.

2. El montaje y mecanización completa de los equipos que intervengan será realizado totalmente por la empresa licitadora.

3. En la oferta, se hará constar el precio total por el que se compromete a realizar la obra y el tanto por ciento de baja que supone este precio en relación con un importe límite si este se hubiera fijado.

4. La obra se realizará bajo la dirección técnica de un Ingeniero Superior de Telecomunicación, auxiliado por el número de Ingenieros Técnicos y Programadores que se estime preciso para el desarrollo de la misma.

5. Aparte del Ingeniero Director, el contratista tendrá derecho a contratar al resto del personal, pudiendo ceder esta prerrogativa a favor del Ingeniero Director, quien no estará obligado a aceptarla.

6. El contratista tiene derecho a sacar copias a su costa de los planos, pliego de condiciones y presupuestos. El Ingeniero autor del proyecto autorizará con su firma las copias solicitadas por el contratista después de confrontarlas.

7. Se abonará al contratista la obra que realmente ejecute con sujeción al proyecto que sirvió de base para la contratación, a las modificaciones autorizadas por la superioridad o a las órdenes que con arreglo a sus facultades le hayan comunicado por escrito al Ingeniero Director de obras siempre que dicha obra se haya ajustado a los preceptos de los pliegos de condiciones, con arreglo a los cuales, se harán las modificaciones y la valoración de las diversas unidades sin que el importe total pueda exceder de los presupuestos aprobados. Por consiguiente, el número de unidades que se consignan en el proyecto o en el presupuesto, no podrá servirle de fundamento para entablar reclamaciones de ninguna clase, salvo en los casos de rescisión.

8. Tanto en las certificaciones de obras como en la liquidación final, se abonarán los trabajos realizados por el contratista a los precios de ejecución material que figuran en el presupuesto para cada unidad de la obra.

9. Si excepcionalmente se hubiera ejecutado algún trabajo que no se ajustase a las condiciones de la contrata pero que sin embargo es admisible a juicio del Ingeniero Director de obras, se dará conocimiento a la Dirección, proponiendo a la vez la rebaja de precios que el Ingeniero estime justa y si la Dirección resolviera aceptar la obra, quedará el contratista obligado a conformarse con la rebaja acordada.

10. Cuando se juzgue necesario emplear materiales o ejecutar obras que no figuren en el presupuesto de la contrata, se evaluará su importe a los precios asignados a otras obras o materiales análogos si los hubiere y cuando no, se discutirán entre el Ingeniero Director y el contratista, sometiéndolos a la aprobación de la Dirección. Los nuevos precios convenidos por uno u otro procedimiento, se sujetarán siempre al establecido en el punto anterior.

11. Cuando el contratista, con autorización del Ingeniero Director de obras, emplee materiales de calidad más elevada o de mayores dimensiones de lo estipulado en el proyecto, o sustituya una clase de fabricación por otra que tenga asignado mayor precio o ejecute con mayores dimensiones cualquier otra parte de las obras, o en general, introduzca en ellas cualquier modificación que sea beneficiosa a juicio del Ingeniero Director de obras, no tendrá derecho sin embargo, sino a lo que le correspondería si hubiera realizado la obra con estricta sujeción a lo proyectado y contratado.

12. Las cantidades calculadas para obras accesorias, aunque figuren por partida alzada en el presupuesto final (general), no serán abonadas sino a los precios de la contrata, según las condiciones de la misma y los proyectos particulares que para ellas se formen, o en su defecto, por lo que resulte de su medición final.

13. El contratista queda obligado a abonar al Ingeniero autor del proyecto y director de obras así como a los Ingenieros Técnicos, el importe de sus respectivos honorarios facultativos por formación del proyecto, dirección técnica y administración en su caso, con arreglo a las tarifas y honorarios vigentes.

14. Concluida la ejecución de la obra, será reconocida por el Ingeniero Director que a tal efecto designe la empresa.

15. La garantía definitiva será del 4% del presupuesto y la provisional del 2%.

16. La forma de pago será por certificaciones mensuales de la obra ejecutada, de acuerdo con los precios del presupuesto, deducida la baja si la hubiera.

17. La fecha de comienzo de las obras será a partir de los 15 días naturales del replanteo oficial de las mismas y la definitiva, al año de haber ejecutado la provisional, procediéndose si no existe reclamación alguna, a la reclamación de la fianza.

18. Si el contratista al efectuar el replanteo, observase algún error en el proyecto, deberá comunicarlo en el plazo de quince días al Ingeniero Director de obras, pues transcurrido ese plazo será responsable de la exactitud del proyecto.

19. El contratista está obligado a designar una persona responsable que se entenderá con el Ingeniero Director de obras, o con el delegado que éste designe, para todo relacionado con ella. Al ser el Ingeniero Director de obras el que interpreta el proyecto, el contratista deberá consultarle cualquier duda que surja en su realización.

20. Durante la realización de la obra, se girarán visitas de inspección por personal facultativo de la empresa cliente, para hacer las comprobaciones que se crean oportunas. Es

obligación del contratista, la conservación de la obra ya ejecutada hasta la recepción de la misma, por lo que el deterioro parcial o total de ella, aunque sea por agentes atmosféricos u otras causas, deberá ser reparado o reconstruido por su cuenta.

21. El contratista, deberá realizar la obra en el plazo mencionado a partir de la fecha del contrato, incurriendo en multa, por retraso de la ejecución siempre que éste no sea debido a causas de fuerza mayor. A la terminación de la obra, se hará una recepción provisional previo reconocimiento y examen por la dirección técnica, el depositario de efectos, el interventor y el jefe de servicio o un representante, estampando su conformidad el contratista.

22. Hecha la recepción provisional, se certificará al contratista el resto de la obra, reservándose la administración el importe de los gastos de conservación de la misma hasta su recepción definitiva y la fianza durante el tiempo señalado como plazo de garantía. La recepción definitiva se hará en las mismas condiciones que la provisional, extendiéndose el acta correspondiente. El Director Técnico propondrá a la Junta Económica la devolución de la fianza al contratista de acuerdo con las condiciones económicas legales establecidas.

23. Las tarifas para la determinación de honorarios, reguladas por orden de la Presidencia del Gobierno el 19 de Octubre de 1961, se aplicarán sobre el denominado en la actualidad “Presupuesto de Ejecución de Contrata” y anteriormente llamado ”Presupuesto de Ejecución Material” que hoy designa otro concepto.

Condiciones particulares

La empresa consultora, que ha desarrollado el presente proyecto, lo entregará a la empresa cliente bajo las condiciones generales ya formuladas, debiendo añadirse las siguientes condiciones particulares:

1. La propiedad intelectual de los procesos descritos y analizados en el presente trabajo, pertenece por entero a la empresa consultora representada por el Ingeniero Director del Proyecto.

2. La empresa consultora se reserva el derecho a la utilización total o parcial de los resultados de la investigación realizada para desarrollar el siguiente proyecto, bien para su publicación o bien para su uso en trabajos o proyectos posteriores, para la misma empresa cliente o para otra.

3. Cualquier tipo de reproducción aparte de las reseñadas en las condiciones generales, bien sea para uso particular de la empresa cliente, o para cualquier otra aplicación, contará con autorización expresa y por escrito del Ingeniero Director del Proyecto, que actuará en representación de la empresa consultora.

4. En la autorización se ha de hacer constar la aplicación a que se destinan sus reproducciones así como su cantidad.

5. En todas las reproducciones se indicará su procedencia, explicitando el nombre del proyecto, nombre del Ingeniero Director y de la empresa consultora.

6. Si el proyecto pasa la etapa de desarrollo, cualquier modificación que se realice sobre él, deberá ser notificada al Ingeniero Director del Proyecto y a criterio de éste, la empresa consultora decidirá aceptar o no la modificación propuesta.

7. Si la modificación se acepta, la empresa consultora se hará responsable al mismo nivel que el proyecto inicial del que resulta el añadirla.

8. Si la modificación no es aceptada, por el contrario, la empresa consultora declinará toda responsabilidad que se derive de la aplicación o influencia de la misma.

9. Si la empresa cliente decide desarrollar industrialmente uno o varios productos en los que resulte parcial o totalmente aplicable el estudio de este proyecto, deberá comunicarlo a la empresa consultora.

10. La empresa consultora no se responsabiliza de los efectos laterales que se puedan producir en el momento en que se utilice la herramienta objeto del presente proyecto para la realización de otras aplicaciones.

11. La empresa consultora tendrá prioridad respecto a otras en la elaboración de los proyectos auxiliares que fuese necesario desarrollar para dicha aplicación industrial, siempre que no haga explícita renuncia a este hecho. En este caso, deberá autorizar expresamente los proyectos presentados por otros.

12. El Ingeniero Director del presente proyecto, será el responsable de la dirección de la aplicación industrial siempre que la empresa consultora lo estime oportuno. En caso contrario, la persona designada deberá contar con la autorización del mismo, quien delegará en él las responsabilidades que ostente.