ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA DE TELECOMUNICACIÓN
UNIVERSIDAD POLITÉCNICA DE CARTAGENA

Proyecto Fin de Carrera

Distribución de contenidos multimedia para publicidad dinámica: un caso práctico

AUTOR: Antonio Miñarro González
DIRECTORA: Dra. Dª María Dolores Cano Baños
Marzo / 2013

INDICE

FIGURAS Y TABLAS

1. Introducción

1.1 Planteamiento inicial del Proyecto

El plan inicial consiste en crear un programa que permita distribuir contenido multimedia por la red, esto es, poder mandar vídeos, fotos al servidor para que éste los procese y envíe a los clientes, que estarán escuchando, para mostrar la información. La distribución de contenidos a través de Internet es la nueva forma de poner accesibles contenidos multimedia y preferentemente audiovisuales, aprovechando las ventajas de acceso universal y los costes reducidos de distribución.

Como uso más común de este tipo de programas tenemos la llamada publicidad digital dinámica, que consiste en mostrar en dispositivos digitales como pantallas o proyectores la información que genera un servidor a partir de los contenidos multimedia introducidos en éste. Este sistema ofrece una flexible e interactiva plataforma de comunicación para promover, informar, educar y entretener a una audiencia definida en un momento y lugar concreto de una forma impactante, atractiva y rentable. Son fáciles de utilizar y se pueden actualizar de forma remota, de forma que los vídeos generados puedan cambiarse con gran facilidad.

1.2 Objetivos del Proyecto

Partiendo de la base del planteamiento inicial se pueden definir una serie de objetivos que el Proyecto tiene que cumplir para su correcta consecución, y son los siguientes:

- Creación de un programa servidor en lenguaje de programación Java que permita recibir fotos o vídeos, que los procese y los una en un único vídeo resultante, el cual será enviado a los clientes.

- Creación de un programa cliente en lenguaje de programación Java que se mantendrá a la espera de que el servidor genere y le envíe el vídeo resultante y muestre esa información.

- Creación de una interfaz web en lenguaje de programación PHP, a la que accederá el administrador del sistema, de tal forma que tenga control absoluto sobre toda la información que contenga, pudiendo actualizar esta información, ya sea agregando archivos o borrándolos para crear futuros vídeos.

1.3 Resumen de la memoria

La memoria del Proyecto está compuesta de los siguientes capítulos:

- Introducción: Se explica brevemente el planteamiento inicial del proyecto, así como sus objetivos principales.
- Soluciones comerciales a la publicidad dinámica: En este apartado viene documentados todos los programas similares que hay actualmente en el mercado.
- Descripción del programa propuesto: Este capítulo está dedicado a la descripción global del proyecto a nivel de funciones apoyado con diagramas UML.
- Ejemplos de uso: Breve descripción de los usos más comunes para este tipo de programas.
- Conclusiones: Comentario final del autor resumiendo sus impresiones y visiones de futuro durante la realización de este proyecto final de carrera.
- Anexo: Aquí se muestra todo el código necesario para el correcto funcionamiento del programa, así como un breve tutorial de uso.
- Bibliografía: Resumen de la bibliografía usada y revisada durante la realización total del proyecto final de carrera.

2. Publicidad dinámica: Soluciones comerciales y comparativa

2.1 Publicidad dinámica digital

La publicidad dinámica digital tiene una gran repercusión en cualquier negocio para promocionar sus productos o mostrar información de interés, llamando la atención del cliente, cautivando al consumidor con contenidos dinámicos y mejorando la comunicación entre empresa y consumidor.

La publicidad digital dinámica se aplica con resultados muy positivos en hostelería, detallistas, centros comerciales, banca, ferias, exposiciones, hospitales, aeropuertos, etc., siempre que cumpla la función de comunicar un mensaje con un objetivo concreto, por ejemplo: informar, publicitar, formar, etc. Hoy en día los sistemas publicitarios basados en este servicio representan el medio más flexible, con más posibilidades de segmentación de la información y ubicados en el mejor punto posible de la decisión de una compra. El éxito de este servicio se basa en el gran impacto debido a las dimensiones de las pantallas y a que están instaladas en zonas con gran afluencia de personas.

La tecnología permite una planificación exacta y más eficaz de campañas publicitarias, ya que el contenido puede ser adaptado al grupo objetivo. Además, la combinación de contenidos interesantes (noticias, agenda local, vídeos, anuncios) garantiza una alta atención y aceptación por parte de los consumidores.

Un ejemplo típico sería una red de pantallas para franquicias, como por ejemplo un supermercado. Desde la central se realiza la gestión y la distribución del contenido que sale en las pantallas de cada sucursal. Todas las pantallas puede mostrar el mismo contenido o distintos. En comparación con la cartelería tradicional el coste económico a largo plazo podría ser mejor, ya que se ahorra en impresión de papel, aunque por otro lado se produce un consumo constante de electricidad para su funcionamiento. Una ventaja evidente es que el sistema de publicidad dinámica digital permite reaccionar con más flexibilidad a las necesidades de cada negocio.

Para entender mejor como funcionan estos programas, así como el programa propuesto en este proyecto final, se muestra un esquema (véase Fig. 1) donde se detallan las claves más importantes que todo sistema dedicado a la publicidad digital dinámica atiende para su correcto funcionamiento y que garantice las necesidades primordiales de este sistema. En primer lugar, un software que facilite la gestión y distribución de los contenidos de forma centralizada. También importante el hardware con el que se trabaja, siendo indispensable para hacer visible el sistema. En tercer lugar hay que definir la estrategia del canal a seguir, donde dirigir los recursos para optimizar el sistema, sus soportes y el lenguaje a usar. Por último la creación de contenidos, con su planificación de forma centralizada según el objetivo final deseado.

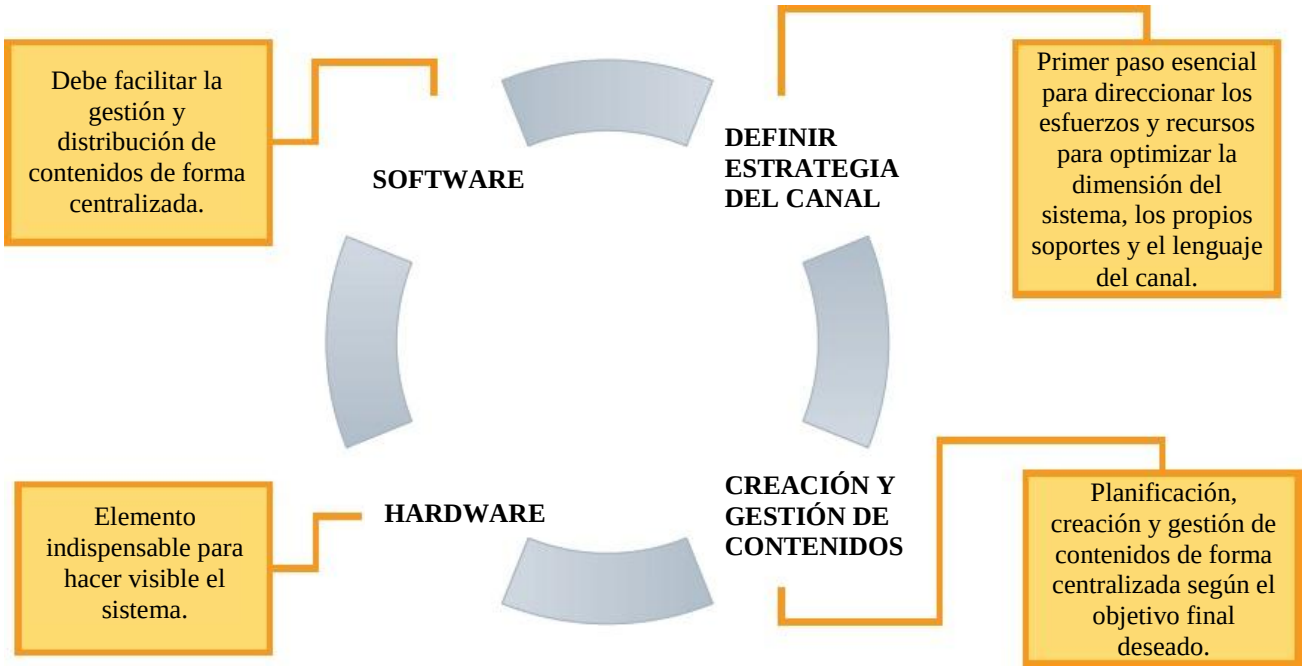

Figura 1. Claves importantes para sistema basado en publicidad dinámica digital.

A continuación se describen ejemplos de programas de publicidad digital dinámica disponibles en el mercado.

2.2 DynaCOM

Dynacom [DynaCOM13] es una empresa ubicada en Barcelona (España) dedicada a la publicidad dinámica digital mediante un sistema integrado que permite mostrar contenidos multimedia en pantallas o videoproyectores ubicados remotamente creando un canal de comunicación propio.

Ante la diversidad de canales de comunicación desde sencillos tablones de anuncios hasta canales de televisión, pasando por vallas publicitarias, periódicos, carteles y escaparates, cada día es más complicado que el mensaje alcance al usuario y le impacte. Entonces, ¿qué podemos hacer para que los consumidores reciban el mensaje justo en el momento en el que están en disposición de comprar? La respuesta es la comunicación digital, un canal que presenta dos ventajas importantes:

- La información no permanece estática, ya que el usuario que está expuesto podrá ver distintos mensajes que podrán cambiar dinámicamente.

- Esta programación de mensajes puede ser totalmente automática y programada previamente que podrán cambiar dinámicamente.

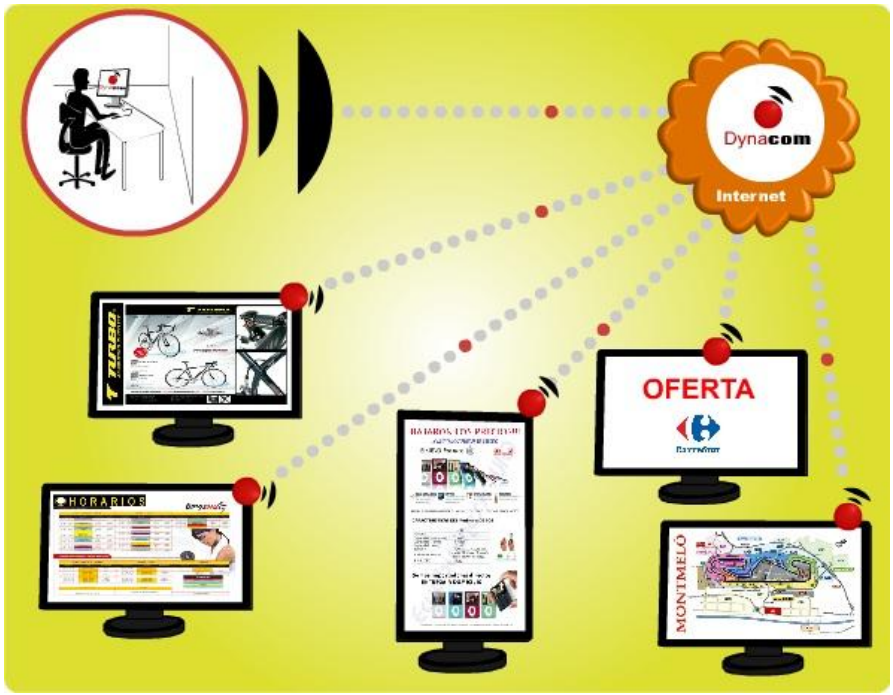

Figura 2. Representación programa de Dynacom

Para el sistema que ofrece Dynacom se necesita tantas pantallas o equipos de visualización como se deseen, desde un monitor de ordenador hasta sistemas multipantalla, videoproyectores o pantallas de plasma y de conexión a Internet para la gestión remota y centralizada de los contenidos mostrados en las diferentes pantallas (véase Fig. 2). La funcionalidad es fácil y cómoda, no necesita de software adicional y consta de tres pasos:

a) **Diseño de la plantilla**. Se genera mediante un software propio de la empresa, que trabaja en un entorno gráfico intuitivo al alcance de cualquier usuario. Podrá contener vídeo, texto, imágenes, páginas web, animaciones flash, audio, etc., para que el usuario pueda combinarlos de manera libre.

b) **Envío de la plantilla**. Se envía a los equipos de visualización de dos formas: de manera manual mediante software de la empresa, correo electrónico, web, FTP o automáticamente mediante servidor web (HTML, ASP, PHP).

c) **Presentación del contenido en el visor**. Muestra la plantilla diseñada anteriormente con la información suministrada hasta el próximo cambio.

Entre los contenidos soportados por el programa de la empresa Dynacom, podemos agruparlos en la siguiente tabla (Tabla 1).

Alguna de las aplicaciones de la compañía Dynacom usando este programa que podemos destacar es la información que se muestra al público en general (véase ejemplo en la Fig. 3). En Ayuntamientos, museos, puntos de venta al cliente o en empresas, siempre es útil mostrar a los visitantes la información de las áreas a las que puedan acceder. Además se puede aprovechar este mismo canal de comunicación para emitir publicidad de servicios ofertados del propio centro o emitir vídeos informativos sobre el uso de las instalaciones o servicios.

2.3 Visionamic

Visionamic [Visionamic13] es una empresa española situada en Barcelona (España) que ofrece soluciones de publicidad dinámica digital para puntos de venta con un software de gestión de contenidos muy fáciles de utilizar, fiables y totalmente actualizables de forma remota.

Visionamic permite la creación, distribución y publicación de contenidos publicitarios, promocionales y/o informativos para una visualización dinámica y en tiempo real, en uno o múltiples puntos de venta, para una audiencia determinada y para ser mostrados en lugares estratégicos.

Tabla 1. Formatos soportados por Dynacom

Imágenes	Fotografías y ficheros gráficos en formato jpg, bmp, gif y png.
Vídeo	Clips de vídeo grabado en formato mpeg, avi y wmv.
Entrada de video en directo	Los equipos disponen de una entrada de video analógico para poder integrar en la pantalla emisiones en directo como Canal+, Ono, TDT o incluso imágenes en directo desde una webcam.
Audio	Música o ficheros de audio en formato .wma y .mp3
Páginas WEB	Páginas HTML, ASP, PHP, etc., servidas de modo local o remotamente desde un servidor.
Power Point	Presentaciones de Power Point en formato nativo (sin necesidad de tener que convertirlas)
Flash	Películas de Flash.
Texto	Texto con cualquier fuente, tamaño y color y con la posibilidad de movimiento.
Reloj	Muestra un reloj en formato analógico o digital con la hora actual.

Figura 3. Información al público en general

Los contenidos dinámicos se adaptan perfectamente a la imagen estética y corporativa ya que el software de Visionamic dispone de más de 150 plantillas disponibles, así como la disponibilidad de cualquier agencia o departamento de Marketing o Comunicación para que genere sus propios fondos o contenidos, integrándolos a cualquier espacio. Las ventajas que Visionamic ofrece con su software son las siguientes:

- Permite llamar la atención y comunicar aspectos relevantes de los productos y servicios a un público objetivo muy segmentado.
- Mostrar el mensaje en el lugar adecuado y en la franja horaria idónea.
- Reducción de costes de impresión en cartelería.

Visionamic es ideal para empresas con diferentes puntos de venta. La agilidad de su sistema permite promocionar cualquier producto en 5 minutos, simultáneamente en todos los puntos de venta, y esto se puede ver en el modo en el que funciona.

Visionamic incorpora un administrador online para el sencillo control remoto de las configuraciones de los equipos. A través de Internet puede controlar los contenidos de las diferentes pantallas. La actualización es inmediata.

En 3 sencillos pasos se obtiene el resultado esperado:

(a) **Resalte su producto**: escoja los productos que desee promocionar, suba las imágenes, vídeos y textos que se visualizará en pantalla (Fig. 4).

(b) **Adapte su contenido**: complete su plantilla que más se adapte a su producto en función de su público con mensajes personalizados y elementos dinámicos como el reloj, ideal para gimnasios que publiquen el programa del día (Fig. 5).

(c) **Presentación al cliente**: Publique el mensaje y las ventas se verá potenciadas (Fig. 6).

Figura 4. Producto para promocionar

Figura 5. Personalización de plantilla

Figura 6. Exposición publicidad dinámica

Con el software de publicidad digital dinámica que Visionamic ofrece a sus clientes podrá transmitir promociones, ventajas del producto, servicios o cualquier tipo de información. Los sectores en donde la presencia de este software destaca son:

- **Comercio:** tiendas, grandes superficies, centros comerciales, boutiques, bazares.
- **Educación:** colegios, institutos, universidades y bibliotecas.
- **Sanidad:** centros de salud y hospitales.
- **Servicios financieros:** bancos y cajas de ahorro.
- **Transporte y turismo:** aeropuertos, agencias de viaje y transporte público.
- **Hostelería:** bares, restaurantes, hoteles.
- **Ocio:** teatros, cines, estadios, museos, ferias, exposiciones, eventos, discotecas.
- **Gobierno:** administración local, sedes de la administración central y autonómica.

2.4 ASVideo

ASVideo [ASVideo13] es una empresa situada en Madrid (España) que lleva más de 20 años ofreciendo a sus clientes proyectos desde producción audiovisual hasta las más recientes y crecientes cartelería digital (publicidad dinámica digital).

La publicidad dinámica digital que ofrece ASVideo está formada por un conjunto de tecnologías de telecomunicaciones y de aplicaciones informáticas que permiten la creación, distribución y publicación de contenidos multimedia informativos y publicitarios mediante dispositivos electrónicos (carteles digitales). Todo ello de una manera dinámica, actual y atractiva que genera impactos visuales y capta la atención en cualquier espacio donde haya afluencia de personas, recepción de empresas, salas de espera, ferias, congresos, cursos, eventos, etc.

La cartelería digital interactiva es una de las nuevas vías de comunicación completamente dinámica, medible y fiable que proporciona una imagen corporativa moderna y avanzada. Capta la atención del público gracias al gran formato de los terminales táctiles interactivos, pudiendo acercar la información y la publicidad de manera ágil e inmediata.

Todo el contenido del terminal es actualizable de manera centralizada, con posibilidades de conectarse al sistema interactivo en cualquier momento. Como herramienta audiovisual, puede funcionar como información general del Evento (temas, horarios, seminarios, ponencias…), información del entorno (mapa de situación, ocio, restaurantes…), información de actualidad (noticias, el tiempo), juegos interactivos (de esta forma se muestra al cliente la información que se desea que capte, aparte de tener un papel participativo en los mismos).

Para ello, ASVideo se basa en terminales con displays digitales de plasma o TFT/LCD de gran formato, 42", 50" o superiores, que se controlan y actualizan remotamente. Puede ser planificado siguiendo una determinada parrilla horaria en la cual se especifica para cada uno de los carteles, el momento de inicio, la duración de los contenidos a publicar y a los clientes que va dirigido.

El sistema que ofrecen aporta interacción, agilidad, ahorro en costes (papel, impresión), innovación con las pantallas, haciendo frente a la cartelería tradicional y otros medios de comunicación clásicos que no ofrecen este impacto ni estas prestaciones. Entre algunos de los proyectos realizados por ASVideo (véase Fig. 7) usando su cartelería digital dinámica se destacan los siguientes:

- **Movie World**, que en 2004 proporcionó a Warner un sistema de terminales con información del Parque Warner Bros sobre servicios y tiempos de espera.
- **Universidad Politécnica de Madrid**, que ofreció un canal de información dinámica interactiva sobre servicios de la propia UPM en 2005.
- **Metro de Valencia**, instalando un canal de información dinámica con publicidad de utilidad y entretenimiento al viajero.

- **Banesto TV**, que en 2007 creó un canal de información dinámica para Banesto que se compone de una proyección y un terminal interactivo de 42”.

- **Aldeasa**, uno de sus últimos proyectos que ofreció un sistema de cartelería digital para los puntos promocionales, cajas e información de la tienda en el aeropuerto de Palma de Mallorca.

Figura 7. Ilustraciones de la publicidad digital dinámica de ASVideo

2.5 Qualitynet

Qualitynet [QualityNET13] es una empresa situada en Valencia (España) que dispone de un equipo de profesionales con experiencia a cada ámbito que concierne a un proyecto en Internet, consiguiendo optimizar tiempo (del cliente) y el dinero invertido de forma que se obtenga el máximo rendimiento del proyecto con el fin de rentabilizar dicha inversión en el mejor tiempo posible.

El auge de Internet y el desarrollo de las nuevas tecnologías de la información someten al mundo de la publicidad a una constante renovación y actualización. Un nuevo concepto de publicidad dinámica exige nuevos canales de comunicación. Es así, que Qualitynet ha desarrollado una nueva plataforma que utiliza Internet para hacer llegar los mensajes a su público objetivo (véase como ejemplo la Fig. 8). Dichos mensajes son visualizados en monitores de televisión, pantallas LCD (o plasma) de grandes dimensiones y su gestión se realiza mediante un navegador. El contenido del mensaje multimedia se elabora a modo de parrilla de programación y permite la emisión de vídeos en directo o diferido.

Con la herramienta que ofrece Qualitynet, se puede conseguir canalizar la información desde el punto de producción al punto de venta (véase Fig. 9). También crea y distribuye contenido multimedia, cualquiera que sea su formato a través de pantallas geográficamente dispersas. Para ellos utiliza dos sistemas independientes entre sí: la zona privada y el sistema de difusión.

Figura 8. Ejemplo de publicidad digital

- **<u>Zona privada</u>:** Una plataforma web de acceso restringido y gestionada directamente por el usuario, le permitirá asignar cualquier tipo de contenido multimedia. En este espacio podrá aplicar y organizar la información que desee enviar a su público. Podrá decidir el contenido de la programación que, posteriormente, se emitirá en las pantallas de televisión. En esta zona podrá subir imágenes, archivos flash o pequeños vídeos; crear secciones de una duración determinada con contenidos audiovisuales o archivos flash, permitiendo almacenar una gran cantidad de información; programar dichas secciones, decidiendo el día y la hora que se emitirá, control de emisión.
- **<u>Sistemas de difusión</u>:** Le permite asignar las presentaciones a los PCs o grupos de PCs que elija. Descargar de modo sencillo las presentaciones, junto con información para su reproducción en la red de monitores. Para que funcione, sólo es necesario disponer de conexión a Internet.

Además, puede programar las presentaciones por días de la semana, por meses, por franjas horarias, etc. Puede elaborar planes personalizados que optimicen el impacto de sus campañas.

Qualitynet le permite gestionar y distribuir sus contenidos publicitarios de la manera más sencilla, sin mayor requisito que disponer de conexión a Internet. Además, con la plataforma que le ofrece Qualitynet puede lograr un alto grado de interactividad y se adaptan según las necesidades propias de cada cliente con su sistema de gestión de publicidad dinámica sin ningún tipo de limitación.

La publicidad dinámica incrementa las ventas en doble proporción que la estática. Esta conclusión es el resultado de los estudios realizados sobre el efecto de la publicidad dinámica en los consumidores. Datos facilitados por la Asociación Global de Marketing revelan que los displays dinámicos incrementan las ventas en un 83% mientras que los estáticos lo hacen en un 39%. La publicidad dinámica revoluciona de este modo el mundo de la comunicación. Algunas de las curiosidades de la publicidad dinámica es que genera 7 veces más atención que la publicidad estática, incrementa las ventas entre un 19% y 36% y aumenta las ventas de artículos de impulso en un 162%. Además, el 75% de las decisiones de compra se producen en el punto de venta.

Figura 9. Estructura publicidad dinámica

Las principales características del sistema de publicidad dinámica digital de Qualitynet son las siguientes:

- **Segmentación**: La flexibilidad que tiene le permite segmentar, filtrar y seleccionar objetivos de campaña, identificando cada una de las pantallas asociadas al cliente. Todo ello, atendiendo a parámetros predefinidos como zona

geográfica, rangos de tiempo (días de la semana o franjas horarias), frecuencias de impacto, clasificaciones tipo creadas por el administrador, etc.

- **Plugins**: Su sistema de gestión permite integrar una serie de plugins estándar a sus contenidos, incrementando así el interés por la información mostrada en sus media players.

- **Flexibilidad**: Permite trabajar con múltiples formatos (imagen fija, vídeo, animación u otros contenidos) y adaptar la presentación de medios a cualquier tamaño de pantalla u orientación.

- **Hosting**: No requiere inversión por parte del cliente en servidores ni en mantenimiento de equipos remotos, la empresa se encarga de mantener los contenidos multimedia con total seguridad y disponibilidad todos los días del año. Por tanto, tan solo es necesario disponer de una conexión ADSL estándar.

- **Creación de contenidos**: Qualitynet posee una productora de vídeo propia, formada por un grupo de profesionales con más de 20 años de dedicación, que garantizan la excelencia de sus producciones, es por eso que ellos mismos realizan el proyecto íntegro, desde su aspecto institucional, imagen corporativa, creación web hasta la realización de los anuncios de todos los productos, servicios y comercios que incluyan el proyecto (Fig. 10).

Figura 10. Parametrización de la publicidad dinámica de Qualitynet

2.6 Comfersa

Comfersa [Comfersa13] es una sociedad mercantil española que gestiona soportes y productos publicitarios, aparcamientos y otros servicios complementarios (atención al viajero, venta de billetes, explotación de locales comerciales…). En la actualidad, sus productos están presentes en 13 estaciones de cercanías en Madrid (Colmenar Viejo, Majadahonda, Torrelodones, Las Rozas, Delicias, Las Margaritas, Méndez Álvaro, Santa Eugenia, La Garena, Parque Polvoranca, Cantoblanco, Torrejón de Ardoz y Parla), así como la estación de Portillo en Zaragoza.

La publicidad dinámica digital de Comfersa es un sistema conformado por 225 pantallas de vídeo repartidas por las estaciones del corredor AVE Madrid-Valencia (véase ejemplo en la Fig. 10), siendo estas dos ciudades sus principales núcleos. Es un sistema basado en el "digital signage" o marketing digital dinámico, esto es, publicidad sobre pantallas, que supone un salto frente a la publicidad estática tradicional. Ofrece anuncios dinámicos en movimiento, con sonido, personalizables e incluso interactivos con el usuario.

Figura 11. Publicidad dinámica en estación de tren.

Comfersa además ofrece un gran despliegue de pantallas, colocadas de manera estratégica y de forma secuencial siguiendo el flujo de viajeros, de tal forma que aunque el viajero vaya andando siempre tenga a la vista una pantalla y vea los anuncios que se exponen para buscar así el máximo impacto posible y el éxito del mensaje.

Figura 12. Secuencia de pantallas de publicidad dinámica.

El sistema, como antes se ha dicho, se compone de 225 pantallas, repartidas de la siguiente forma: 204 pantallas horizontales de 42" y 55" (ejemplo en la Fig. 12), 6 videowalls y 15 pantallas verticales táctiles interactivas. Se emiten anuncios comerciales de 15 a 20 segundos, con un número máximo de 6 anunciantes, para evitar la saturación. Las pantallas emiten 18 horas al día (de 6:00h a 24:00h). Un anuncio se emitirá unas 600 veces al día en una sola pantalla, 3.780.000 veces en todas las pantallas de publicidad dinámica en un mes. Las pantallas verticales, además de emitir vídeo, son soportes táctiles interactivos con información de servicio al viajero: mapa de situación de la estación, hoteles, restaurantes cercanos, tiendas de dentro de la estación, alquiler de coches, etc., así como acceso directo a las webs oficiales de turismo de cada ciudad.

Las ventajas de la publicidad dinámica de Comfersa son:

- Posibilidad de segmentación al máximo detalle: por horas, por estación, e incluso por zonas dentro de una misma estación.
- Alcanza un 90% de recuerdo sugerido en los viajeros.
- Eficacia un 60% superior a los soportes estáticos.
- El aumento de ventas durante una campaña digital es de un 24%.

2.7 Comparativa de programas

Está claro que la publicidad digital dinámica está revolucionando la forma de comunicar a los clientes los diferentes productos ofertados y es por ello que cada vez hay más empresas que se dedican a ello. En esta memoria se hace hincapié en cinco programas que seguramente están entre las más importantes dentro del sector, ofreciendo sus servicios de manera muy clara, sencilla y con la mayor transparencia posible. A continuación, presentaremos una tabla comparativa (Tabla 2) de lo que nos ofrece cada programa.

Tabla 2. Comparativa de programas

	DynaCom	Visionamic	ASVideo	QualityNet	Comfersa
Proporciona la información			✓		✓
Formatos de imagen (jpg, png,...)	✓	✓		✓	
Formatos de vídeo (avi, mov, mpg,...)	✓	✓		✓	
Formatos de Office (doc, xls, ppt,...)	✓	✓			
Formatos de audio (mp3, wav,...)	✓			✓	
Formatos de animación (flv, swf,...)	✓			✓	
Iteración con el cliente (táctil)			✓		✓
Plantilla predefinida	✓	✓			
Plantilla libre	✓			✓	
Configuración de secuencia de publicidad por la empresa o cliente	Empresa	Empresa	Empresa	Empresa	Empresa
Creación de plantilla desde web, software o empresa	Software	Página web	Empresa	Página web	Empresa
Actualización automática de nuevos contenidos	✓	✓	✓	✓	✓

Como podemos observar, hay tres programas muy parecidos entre sí, que ofrecen total libertad al cliente para poder subir sus archivos multimedia para la confección de sus anuncios a mostrar (DynaCOM, Visionamic, y QualityNet). El primero, más versátil, proporciona la posibilidad de crear una plantilla propia o algunas predefinidas, mientras que los otros ofrecen o una o la otra. Con respecto a los formatos que admite cada uno, DynaCOM es el más completo, ya que admite todos los formatos posibles. Visionamic sin embargo es el que menos ofrece, siendo únicamente los formatos de imágenes, vídeos y textos. Y por último QualityNET ofrece todos los formatos, excepto documentos de Microsoft Office. Además de garantizar la actualización automática de nuevos contenidos, DynaCOM ofrece sus servicios mediante un software exclusivo de la empresa, mientras que los otros dos programas lo hacen directamente desde una página web.

De los otros dos programas, ASVideo y Comfersa, hay una clara diferencia con respecto a los otros tres descritos anteriormente. En el primer caso (ASVideo) no dispone de una conexión directa con el cliente, ya que los clientes que contratan los servicios de ASVideo no mandan archivos multimedia al servidor, sino que proponen la información que desean mostrar y es la empresa quien realiza el anuncio, usando sus propias técnicas para su distribución a quienes lo solicitaron. Y en el segundo caso (Comfersa) es un programa de publicidad digital dinámica que usa la empresa para mostrar sus anuncios en puntos estratégicos, buscando tener mayor impacto en los clientes. Todo se gestiona desde la propia empresa, tanto la creación de los anuncios publicitarios como la distribución a todas las pantallas. Entre todas esas pantallas, también se disponen de pantallas táctiles que ofrecen a los usuarios la posibilidad de interactuar con ellas para conseguir diversa información. En particular, Comfersa actúa directamente en todas las estaciones del corredor AVE Madrid-Valencia.

3. Descripción del programa propuesto

Como bien se especifica en los objetivos del proyecto el programa propuesto se ha realizado usando el lenguaje de programación Java, que debe ser capaz de gestionar los múltiples archivos multimedia que se carguen en el sistema, procesarlos de tal forma que generen un vídeo final y su posterior distribución, y la interfaz web con el lenguaje de programación PHP, que será la herramienta con la que se gestionen los ficheros que se han de subir al sistema.

En la parte que concierne al servidor se distingue dos bloques distintos, dependiendo de las exigencias del administrador a la hora de crear el vídeo final a mostrar, ya que pueden ser a partir de vídeos o a partir de fotos. Entre los formatos que admiten cada parte podemos decir que para los vídeos sólo se admite la extensión ".avi", cuyo tamaño no debe exceder los 128 Megabytes por vídeo y para las fotos sólo se admite ".jpg", cuyo tamaño no tiene límite. Además, si hay distintos comercios a los cuales vayan dirigidos diferentes vídeos, desde la interfaz web puedes elegir a quién va distribuir el vídeo final, teniendo también la posibilidad de mandar a todos el mismo.

En la parte del cliente el proceso es bastante sencillo, ya que dispone de una herramienta que sólo hay que arrancar manualmente, introduciendo el tipo de comercio que es, y automáticamente se realizará todo el proceso de recepción de vídeos. Incluye un sistema de preguntar al servidor si hay vídeos nuevos, de tal forma que siempre estará conectado y permite la actualización en tiempo real de nuevos contenidos. Además, siempre estará reproduciendo los vídeos que le lleguen, desde el más nuevo hasta el más antiguo.

Para poder programar en Java, teniendo que manipular diferentes tipos de ficheros multimedia, es necesario el uso de una de sus librerías que se llama Java Media Framework (JMF). La API JMF es una definición del interfaz usado por Java para la utilización de archivos multimedia y su presentación en applets y aplicaciones. Como definición de interfaz, JMF no tiene por qué proporcionar las clases finales que manejan datos multimedia, pero dice cómo los proveedores de dichas clases deben encapsularlas y registrarlas en el sistema.

Sus principales características son:

- Estabilidad debida a que funciona sobre la máquina virtual java (JVM).
- Sencillez, ya que permite, usando unos pocos comandos, realizar complejas tareas multimedia.
- Potencia, permitiendo la manipulación de elementos multimedia de vídeo y/o audio locales (procedentes de la misma máquina en la que se ejecuta el programa), así como la retransmisión en tiempo real del vídeo a través de la red.

JMF nació con el objetivo de poder representar datos multimedia en los programas, pero desde las últimas versiones las capacidades se extienden a todo tipo de tratamiento como la adquisición, procesado y almacenaje de datos multimedia, así como la transmisión y recepción a través de la red mediante el protocolo RTP.

JMF ha sido desarrollada por Sun e IBM y no está incluida en las especificaciones de Java 2 o de la máquina virtual, por lo que es necesario obtener el paquete adicional que contiene el JMF para la plataforma que se esté utilizando.

En todo tratamiento que se pueda hacer con los datos multimedia siempre existen tres pasos (véase Fig. 13):

- Adquisición de datos, ya sea una captura desde un dispositivo físico, lectura de un fichero o recepción desde la red.
- Procesado, que se aplican efectos como filtrado o realces, conversión entre formatos o conversión/descomprensión.
- Salida de datos, cuya presentación será mediante almacenamiento en un fichero o transmisión a través de la red.

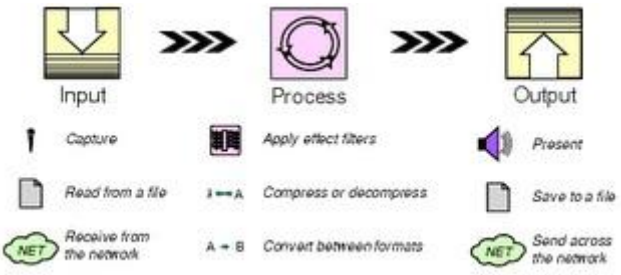

Figura 13. Fases de la API JMF

Para la realización de la interfaz web se ha optado por usar el lenguaje de programación PHP. Es un lenguaje de programación interpretado para HTML, diseñado originalmente para la creación de páginas web dinámicas. Se usa principalmente para la interpretación del lado del servidor, pero actualmente puede ser utilizado desde una interfaz de línea de comandos o en la creación de otros tipos de programas incluyendo aplicaciones con interfaz gráfica.

3.1 Estructura general cliente/servidor

El funcionamiento general del programa (véase Fig. 14) entre cliente y servidor se va a llevar de manera secuencial y dinámica, siendo una comunicación directa y de manera infinita hasta que alguno de las dos partes decida cerrar la conexión.

El servidor estará siempre en funcionamiento, a la espera de nuevos clientes para conectarse al servicio. Cuando un cliente se conecta manda su identificador de comercio y se mantiene a la espera de que se generen los vídeos. Si ya hay vídeos generados, recibirá el último vídeo con su identificador. Tras un tiempo de espera, el cliente pregunta al servidor si hay vídeos nuevos, si no hay volverá a esperar el tiempo de espera y seguirá así hasta que termine la conexión.

A continuación explicamos en detalle cada parte de la conexión cliente/servidor

- **Servidor**: Sencilla clase en java que únicamente habilita en el servidor el puerto por donde escuchará las peticiones de los clientes. Cuando un cliente se conecta, se le abre un hilo independiente donde poder tratar sus peticiones sin que afecte a otros clientes que puedan conectarse.

```
while (true) {

  cliente = servidor.accept();
  start();

}
```

- **AtiendeCliente**: En esta clase es donde se va a interactuar con los clientes que se conecten para enviar los vídeos que se generen en el servidor. En la primera

conexión de un cliente se le mandará el vídeo más reciente que haya de su tipo o del tipo general.

```
DataOutputStream dos = new
DataOutputStream(cliente.getOutputStream());
System.out.println("Preparando para enviar...");
dos.writeUTF(fichero);
enviar();
```

Si aun no hay vídeos lo dejará en espera. Si el cliente es antiguo irá preguntando por si hay vídeos nuevos para él, si no hay, no pasará nada, pero si sí lo hay, se le mandará como si fuera la primera conexión.

```
if (!fichero.equals(pregunta)) {
  System.out.println("Preparando para enviar...");
  dos.writeUTF(fichero);
  enviar();
}
```

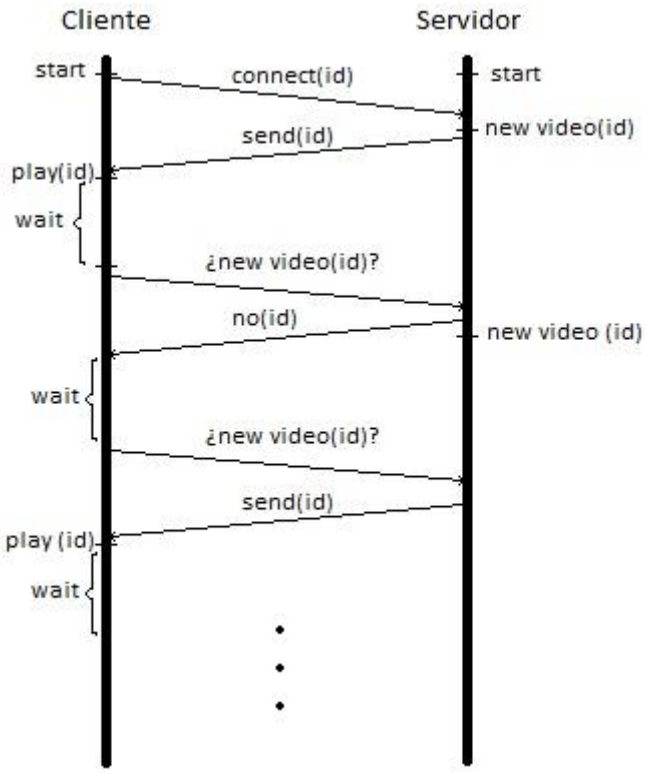

Figura 14. Diagrama de funcionamiento

3.2 Servidor. Vídeo a partir de vídeos

Uno de los dos bloques de los que se hablaba al comenzar el tercer capítulo en el servidor es la creación de un vídeo a partir de varios vídeos. El administrador del sistema puede cargar dos o más vídeos al servicio, puede elegir qué tipo de vídeo final está generando, si va dirigido a un comercio de alimentos, de cosméticos o para todos en general, pudiendo generar el vídeo final que se mostrarán a los clientes correspondientes que estarán escuchando, esperando la información para poder mostrarla.

Para el correcto funcionamiento de esta parte del servidor se han tenido que crear varias clases en Java que describo a continuación:

- **Concatenar.java**: Esta es la clase más extensa que tiene todo el programa, la cual se encarga de unir los vídeos que se le pasan como argumento de entrada en un archivo de salida, que también se le pasa como argumento de entrada. Los vídeos que se van a unir se pasan todos juntos en un "array" de "String" donde están incluidas las rutas de cada vídeo. Esta clase ha sido descargada de la página de Oracle, que lleva por título "Concat.java" en la parte dedicada a JMF donde exponen varios ejemplos para comprender su manejo y utilización. En ella se tiene en cuenta los tipos de archivos de entrada y salida, comprueba si disponen del mismo formato, las mismas características y básicamente verificar que son compatibles para la unión. El proceso que lleva a cabo es el de separar pistas de audio y vídeo, los carga en un "buffer" que finalmente los va guardando en el fichero final señalado.

```
for (j = 0; j < tcs.length; j++) {
  if (tcs[j].getFormat() instanceof AudioFormat) {
    tInfo = new TrackInfo();
    tInfo.idx = j;
    tInfo.tc = tcs[j];
    pInfo[i].tracksByType[AUDIO][aIdx++] = tInfo;
  } else if (tcs[j].getFormat() instanceof VideoFormat) {
    tInfo = new TrackInfo();
    tInfo.idx = j;
    tInfo.tc = tcs[j];
    pInfo[i].tracksByType[VIDEO][vIdx++] = tInfo;
  }
}
int total[] = new int[MEDIA_TYPES];
for (type = AUDIO; type < MEDIA_TYPES; type++) {
```

```
  total[type] = pInfo[0].numTracksByType[type];
}

for (i = 1; i < pInfo.length; i++) {
  for (type = AUDIO; type < MEDIA_TYPES; type++) {
    if (pInfo[i].numTracksByType[type] == total[type])
      total[type] = pInfo[i].numTracksByType[type];
    }
  }
  if (total[AUDIO] > 1 && total[VIDEO] > 1) {
  System.err.println("There is no audio or video tracks to
concatenate.");

  return false;
}

videosTotales = 0;
for (type = AUDIO; type < MEDIA_TYPES; type++)
  videosTotales += total[type];
  for (i = 0; i < pInfo.length; i++) {
    for (type = AUDIO; type < MEDIA_TYPES; type++) {
    for (j = total[type]; j < pInfo[i].numTracksByType[type]; j++) {
     tInfo = pInfo[i].tracksByType[type][j];
     disableTrack(pInfo[i], tInfo);
   }
   pInfo[i].numTracksByType[type] = total[type];
    }
  }


  for (type = AUDIO; type < MEDIA_TYPES; type++) {
    for (i = 0; i < total[type]; i++) {
      if (!tryMatch(pInfo, type, i)) {
        return false;
      }
    }
  }
}
```

El vídeo final generado es de extensión .avi, con lo cual se recomienda que los vídeos de entrada también lo sean para un correcto uso del programa. En caso contrario, la propia clase Concatenar intentará codificar los vídeos de entrada para generarlos en un único vídeo .avi, pero no siempre se asegura el éxito de este proceso por pérdida de información a la hora de separar audio y vídeo.

- **Video.java**: En esta clase se programa el método principal (main) que dirigirá la clase anterior mencionada, configurándola y preparándola para el proceso de unión de los dos o más vídeos que reciba como entrada. Básicamente comprueba que todos los archivos de la carpeta origen sean vídeos, comprueba su extensión y añade sus rutas de acceso a un "array".

```
FilenameFilter filter = new FilenameFilter() {
  public boolean accept(File directorioV, String name) {
    return name.endsWith(".avi") || name.endsWith(".AVI")
    || name.endsWith(".mpg") || name.endsWith(".MPG")
    || name.endsWith(".mov") || name.endsWith(".MOV");
  }
};

File archivos[] = directorioV.listFiles(filter);
String videos[] = new String [archivos.length];
```

También crea el fichero de salida, que junto al "array" de los vídeos de entrada se pasarán como argumentos de entrada a la clase Concatenar para realizar la unión de los vídeos en el archivo final.

A continuación, y para terminar de entender el funcionamiento de esta parte del servidor, se expone el diagrama UML de esta parte del programa (Fig. 15):

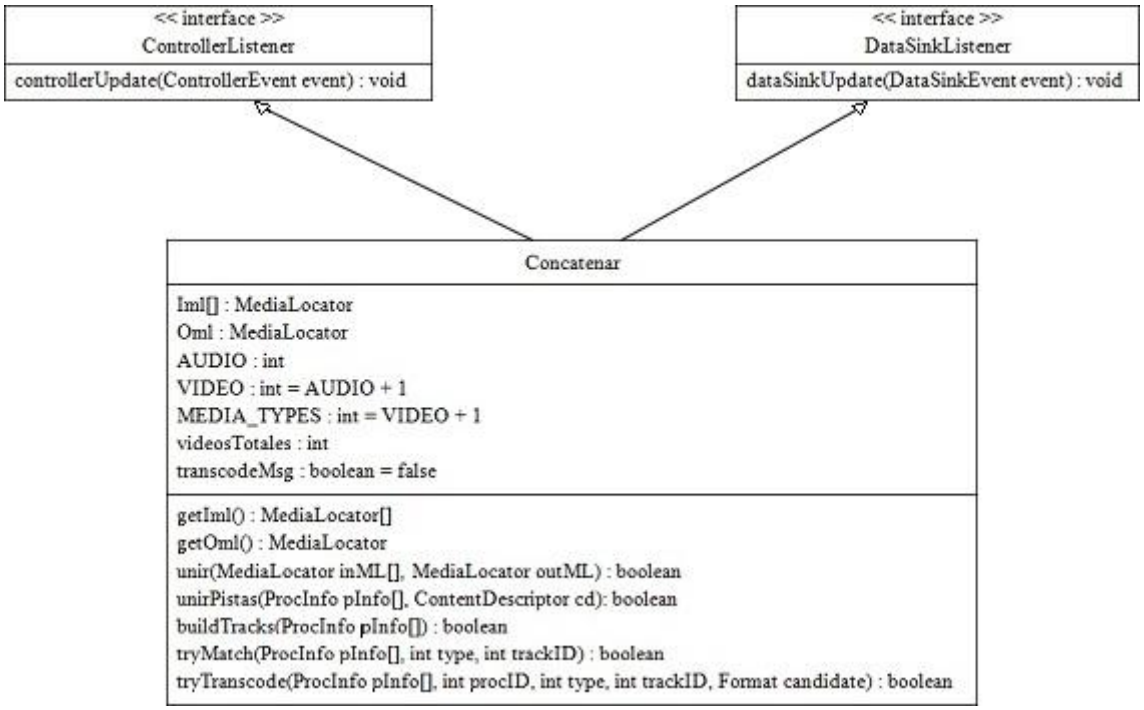

Figura 15. Diagrama UML del servidor creando un vídeo a partir de vídeos

3.3 Servidor. Vídeo a partir de fotos

El segundo bloque en la ejecución del servidor es el referente a la creación de un vídeo a raíz de una serie de fotografías cargadas en el sistema. Al igual como ocurre en el bloque anterior el administrador puede cargar al sistema las fotografías que desee para generar el vídeo, definir también el tipo de vídeo generado para distinto comercio o para todos y además, también podrá definir el tiempo de duración de cada una de las fotografías que se mostrarán en el vídeo final.

Para este apartado, han sido necesarias programar más clases en Java, ya que era necesario tratar las fotografías dependiendo de su tamaño, y su duración durante el vídeo. A continuación los detalles de cada una de las clases:

- **Acciones.java**: En esta clase se han agrupado la mayoría de las funciones con las que trabajará esta parte del servidor para el procesado de las imágenes y su puesta a punto del vídeo. En su primer método guarda todas las fotos de la ruta la cual se cargan las imágenes, comprobando antes que se trate de imágenes (.jpg).

```
FilenameFilter filter = new FilenameFilter() {
  public boolean accept(File dir, String name) {
   return name.endsWith(".jpg") || name.endsWith(".JPG");
  }
};
```

En su segundo método, devuelve un MediaLocator para el archivo que se le pasa como argumento. En el tercero da los formatos oportunos para el vídeo final que se generará. En el cuarto y quinto método generará el proceso que guarda las fotos en un vídeo. Y por último, en el sexto método irá cargando foto a foto de las cargadas en el primer método, de tal forma que irá almacenando un array de bytes con toda su información.

```
raFile = new RandomAccessFile(imageFile, "r");
byte data[] = null;
if (data == null || data.length < raFile.length()) {
  data = new byte[(int) raFile.length()];
}
```

- **ImageDataSource.java**: Es una clase derivada de *PullBufferDataSource* la cual nos permitirá leer ficheros JPG y que nos permite obtener buffers que contienen la información binaria de las imágenes decodificadas una a una. Cada valor que devuelve esta clase es un *PullBufferStream*.

- **ImageSourceStream.java**: Es una clase derivada de *PullBufferStream* la cual recogerá los valores de salida de la clase anterior descrita, y que será atendida por el procesador que convierte el vídeo para cada una de las tramas del vídeo final, donde se irá añadiendo los bytes de cada una de las fotos solicitadas. Cuando se han leído todas las imágenes para generar el vídeo se lanza un evento especial que permite parar el proceso de añadir imágenes al vídeo.

- **RedimensionarImagen.java**: Esta clase tuvo que hacerse debido a las dimensiones de las fotografías que se quieren subir, ya que no es lo mismo mostrar en un vídeo una imagen horizontal que otra vertical. Es por esto, que esta sencilla clase tenemos tres métodos bien diferenciados. El primero de ellos carga la fotografía que se le pase, y la guarda para su procesado. En el segundo diferencia si la imagen es vertical u horizontal sobre un fondo de color negro. Si es horizontal, establecerá los límites marcados por la resolución de pantalla, pero si es vertical, tomando como referencia el alto (determinado por la resolución) el ancho será reducido con el mismo porcentaje de ampliación/reducción que haya tenido el alto mediante una sencilla regla de tres.

```
if (w<h){
  newW = (w*newH)/h;
  g.setRenderingHint(RenderingHints.KEY_INTERPOLATION,
RenderingHints.VALUE_INTERPOLATION_BILINEAR);
  g.drawImage(bufferedImage, (auxW/2)-(newW/2), 0,
(auxW/2)+(newW/2), newH, 0, 0, w, h, null);
  g.dispose();
  return bufim;
}

else {
  g.setRenderingHint(RenderingHints.KEY_INTERPOLATION,
RenderingHints.VALUE_INTERPOLATION_BILINEAR);
  g.drawImage(bufferedImage, 0, 0, newW, newH, 0, 0, w, h,
null);
  g.dispose();
  return bufim;
}
```

Finalmente, el tercer método guarda la imagen nueva para continuar el programa.

- **GenerarVideo.java**: En esta clase se produce y termina el proceso de crear el vídeo final a partir de las fotos cargadas en el sistema. Mediante usos de las clases anteriores, la clase GenerarVideo cargará las imágenes según la clase Acciones y creará el archivo de vídeo final para que con su método "start" comience el proceso que permitirá dar formato al archivo generado y lo guardará en un archivo de película QuickTime (.mov).

```
VideoFormat vFormat = actions.createVideoFormat(
width, height, Format.NOT_SPECIFIED,
Format.byteArray, frameRate);

processor.configure();

processor.setContentDescriptor(new
ContentDescriptor(FileTypeDescriptor.QUICKTIME));
TrackControl tcs[] = processor.getTrackControls();
Format f[] = tcs[0].getSupportedFormats();

processor.realize();

dsink = actions.createDataSink(processor, outML);
dsink.open();
processor.start();
dsink.start();
```

- **Fotos.java**: Esta clase contiene el método principal (main) la cual permitirá controlar las clases anteriores mencionadas. En un primer instante creará la ruta origen de donde se sacarán las fotografías y también la ruta del archivo de vídeo final. Una vez cargue las fotografías, las redimensionará según la resolución de pantalla específico de la máquina y las irá guardando en una ruta diferente para no modificar las imágenes de origen.

```
int ancho, alto;
if (at.getAncho() == 0) {
  ancho = Toolkit.getDefaultToolkit().getScreenSize().width;
  alto = Toolkit.getDefaultToolkit().getScreenSize().height;
}

File fotos[] = directorioI.listFiles();
RedimensionarImagen ri = new RedimensionarImagen();
BufferedImage origen, destino;
```

Finalmente llamará a la clase GenerarVideo con la ruta de donde están las fotos y la ubicación del archivo final para su almacenamiento.

```
gv = new GenerarVideo(video, dirImages.getPath());
gv.start(ancho, alto, frame);
```

Para entender esta parte del servidor donde genera el vídeo a partir de fotos se añade el diagrama UML (Fig. 16):

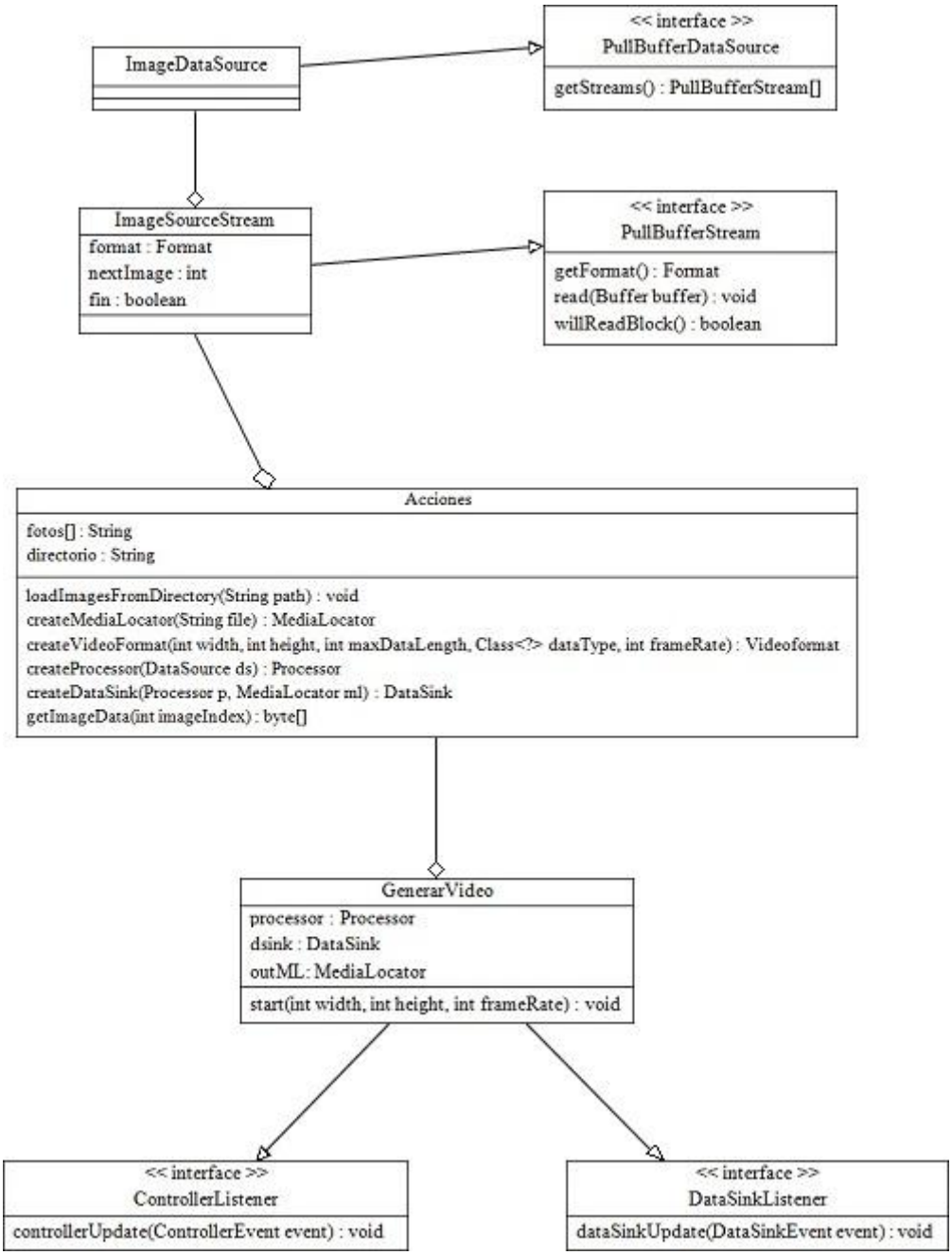

Figura 16. Diagrama UML del servidor que se encarga de generar un video a partir de fotos.

3.4 Servidor. Interfaz web

Una vez programado todo lo que concierne al servidor es hora de programar la interfaz gráfica que permita al administrador del sistema poder manejar de forma clara y sencilla todas las posibilidades que el servicio le ofrece.

Para esta ocasión emplearemos el lenguaje de programación PHP ya que ofrece mejores posibilidades para crear una interfaz web de manera más próxima a las necesidades del administrador, puesto que este lenguaje está orientado al desarrollo de aplicaciones web dinámicas.

La interfaz web (Fig. 17) es de sencilla interpretación, con sólo lo necesario para no complicarlo en exceso. En el bloque de fotos, se muestra la lista de las fotos actualmente cargadas en el servidor junto a los segundos que durará en el vídeo. A la derecha hay dos botones para eliminar la foto o visualizarla; debajo se añade un cuadro de texto con información de interés sobre la carga de los archivos, un desplegable donde marcaremos el tipo de vídeo para quién va dirigido (o a todos) y el botón final que generará el vídeo y enviará a los clientes correspondientes que estén escuchando.

Figura 17. Interfaz web dedicada a las fotos cargadas en el sistema para generar vídeo.

En el bloque de vídeos es prácticamente igual al del bloque de fotos, se ven de igual forma los vídeos cargados en el sistema, con los mismos botones de eliminar o de visualizar (sólo podrá visualizarse si el navegador soporta esa función). También dispone del cuadro de texto donde se muestra información de interés sobre los vídeos que se pueden cargar, el desplegable con los diferentes comercios a quien puede ir dirigido (o a todos) y finalmente el botón que genera el vídeo final (Fig. 18).

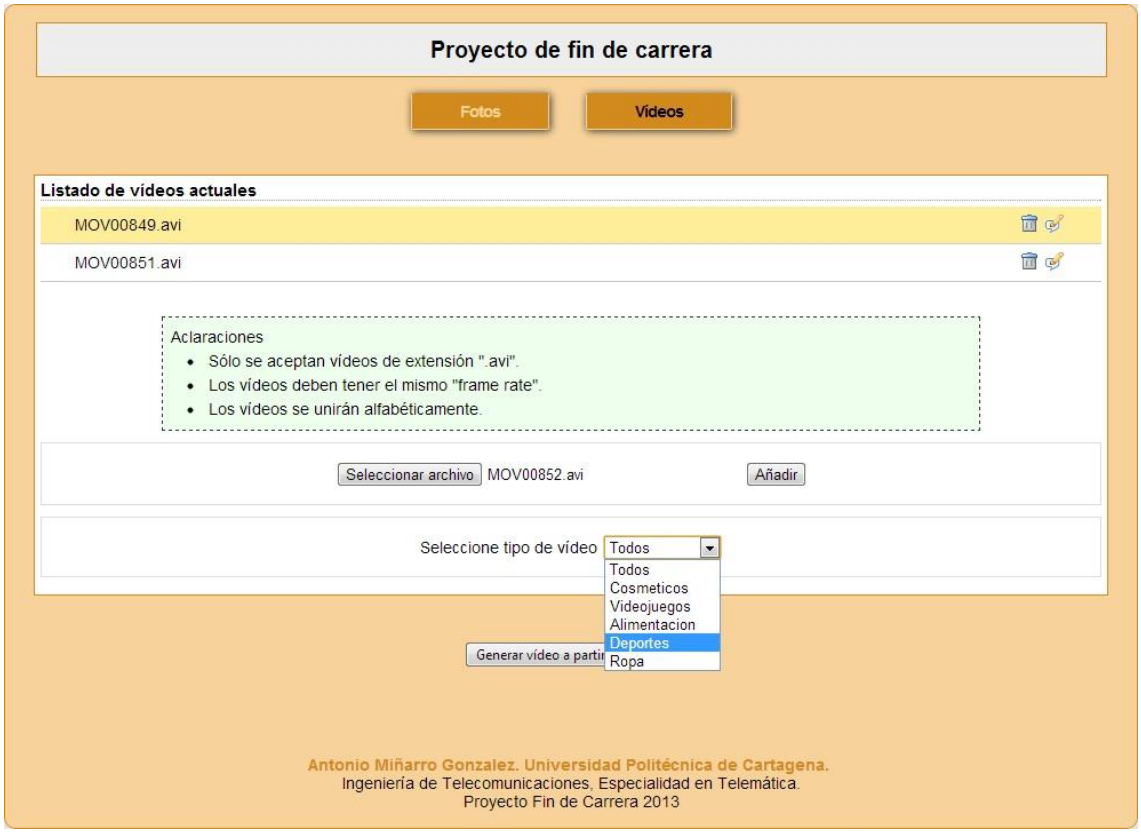

Figura 18. Interfaz web dedicada a los vídeos cargados en el sistema para generar vídeo.

3.5 Cliente

Con todo programado en cuanto al servidor (programa y la interfaz web que lo administra) falta programar el programa cliente y su conexión con el servidor. Los clientes se conectarán al servidor, que estará siempre funcionando, se mantendrán a la espera de que se generen vídeos y los reproducirán.

Cuando los clientes quieren conectarse al servicio, deberán antes especificar qué tipo de comercio son, pues hay que hacer distinción entre los vídeos generados, no es lo mismo que una tienda de videojuegos reciba un vídeo sobre cosméticos. Para ello,

cuando inician la conexión se les muestra un menú donde podrán seleccionar su negocio. De este modo, cuando el servidor genere un vídeo el cliente lo recibirá o no dependiendo de si va dirigido a su comercio o no. Cada vez que un cliente reciba su primer vídeo, éste se almacena para su posterior reproducción de manera indefinida. Si el cliente recibe más vídeos, se cortará la reproducción actual, empezando a reproducir el último vídeo recibido, y se irán reproduciendo desde el más nuevo al más antiguo, también de manera indefinida.

A continuación explicamos en detalle cada parte de la conexión cliente/servidor

- **Cliente**: El cliente se interpreta que será un proyector o una pantalla LCD, donde reproducirá los vídeos que genere el servidor. Al arrancar, se le mostrará un menú (véase ejemplo en la Fig. 19) donde podrá seleccionar el tipo de comercio que es y escribir la dirección IP del servidor. Seguidamente procede a la conexión.

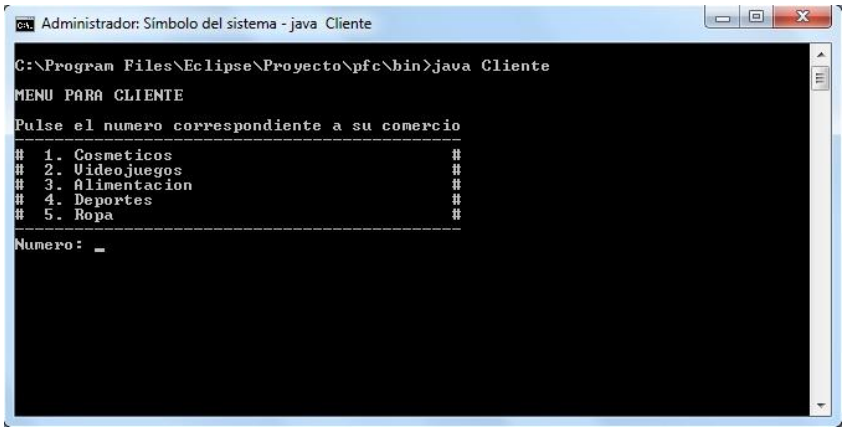

Figura 19. Ilustración del menú para clientes.

La primera vez que se conecta comprueba si existen vídeos generados de su tipo o de general para descargarlo y proceder a la reproducción. Si no hubiera, se mantiene a la espera a que se genere. A partir de este momento el cliente estará reproduciendo el vídeo y cada 30 segundos preguntará al servidor si hay algún vídeo nuevo. De no haber, seguirá reproduciendo, y cuando haya un vídeo nuevo, lo descargará y lo reproducirá y vuelta a empezar.

```
bis = new BufferedInputStream(s.getInputStream());
bos = new BufferedOutputStream(new
FileOutputStream(fichero));
```

```
buffer = new byte[4096];

System.out.print("Recibiendo video");
int x=0;
  while ((n = bis.read(buffer)) != -1){
    bos.write(buffer,0,n);
    if (x%185==0) System.out.print(".");
    x++;
  }
System.out.print(" Recibido!");

Thread.sleep(30000);
```

- **Reproductor**: Esta clase java se usa en la parte del cliente que servirá para poder almacenar los vídeos que el cliente vaya descargando y poder reproducirlos de manera indefinida. Cada vez que se almacena un vídeo, se le extrae la duración de éste para que sirva de guía para esperar entre vídeo y vídeo y no se solapen unos con otros. De manera secuencial, el algoritmo va reproduciendo del vídeo más reciente al más antiguo de manera infinita.

```
for (int i=videos.length-1; i>=0; i--){
  if (videos[i] == null) continue;
  System.out.println("Reproduciendo");

  try {
    Runtime.getRuntime().exec("rundll32
url.dll,FileProtocolHandler  "+videos[i]);
    Thread.sleep(getDuration(videos[i])*1000);
  } catch (Exception e) {
    e.printStackTrace();
  }
  if (i==0) i = videos.length-1;
}
```

4. Conclusión

Siguiendo la línea de trabajo propuesta en el inicio del proyecto, así como en los objetivos del mismo, se puede deducir que todo se ha hecho según lo previsto. Me he documentado bien sobre la API Java Media Framework, la base de todo el programa propuesto y seguidamente me dispuse a programar las distintas clases Java de las que se compone. Enfoqué en un primer instante a realizar el primer bloque del servidor (unión de fotos) y tras su éxito dediqué el tiempo al segundo bloque (unión de vídeos).

Una vez programado el programa encargado de la unión de los archivos multimedia y su distribución, realicé la interfaz web en PHP, sirviéndome de la documentación dada durante la carrera para poder crearla de manera sencilla y clara. Terminada esta parte, ya solo me quedaba programar las clases para los clientes, así como la forma de poder reproducir los vídeos de manera secuencial, reproduciendo del más nuevo al más antiguo.

Como visión de futuro del programa, tiene la ventaja de que no tendría que terminar aquí. Te encuentras en la situación de que se ha completado el proyecto final de carrera, pero que aún podría mejorar, como es el caso de añadir audio a un vídeo de fotografías o también a la inclusión de un nuevo formato final de salida, como una animación en flash. Son objetivos que se podrían plantear a largo plazo.

El proyecto en sí me ha supuesto un reto difícil, sobre todo al principio, mayormente por no tener los conocimientos adecuados para la programación en Java de este tipo de programas, ya que durante la carrera no se ofrece ese tipo de lenguaje. Todo lo relacionado con el tratamiento de archivos multimedia con Java ha sido completamente nuevo para mí, teniendo que buscar más información de lo habitual para poder acercarme más a lo que necesitaba hacer.

Pienso además, que el proyecto realizado para la publicidad digital dinámica debe su éxito a la capacidad de generar contenidos nuevos en tiempo real y que tiene realizarse desde un único equipo, donde la captura de archivos, procesado de la información y almacenamiento quede como un sistema centralizado, y que los

dispositivos digitales, como proyectores o pantallas queden al margen y se dediquen exclusivamente a reproducir el vídeo final.

Y ya por último sólo me queda comentar que el resultado obtenido una vez que se ha terminado el proyecto ha superado con creces todo tipo de imaginación que tenía sobre el mismo durante el planteamiento inicial y búsqueda de información para su puesta en marcha.

5. Anexo

5.1 Código del programa

A continuación se va a poner el código de todos los archivos java necesitados para el correcto funcionamiento del programa en el orden que aparecieron explicados en la presente memoria de este proyecto final de carrera.

Servidor.java

```
import java.net.ServerSocket;
import java.net.Socket;

public class Servidor {

	public static void main (String args[]){

		try {

			ServerSocket servidor = new ServerSocket(12345);
			Socket cliente;

			while (true) {

				cliente = servidor.accept();

				System.out.println("Peticion de video de " +
				cliente.getInetAddress());

				AtiendeClientes at = new
				AtiendeClientes(cliente);

				at.start();

			}

		} catch(Exception e){
			e.printStackTrace();
			System.exit(0);
		}
	}
}
```

AtiendeClientes.java

```
import java.io.BufferedInputStream;
import java.io.BufferedOutputStream;
import java.io.DataInputStream;
import java.io.DataOutputStream;
import java.io.File;
import java.io.FileInputStream;
import java.io.FilenameFilter;
import java.net.Socket;
import java.net.SocketException;

public class AtiendeClientes extends Thread {

	private Socket cliente;
	int ancho = 0;
	int alto = 0;
	File archivo;
	String fichero;
	char c = 0;

	public AtiendeClientes (Socket s){
		cliente = s;
	}

	public AtiendeClientes() {}

	public int getAncho(){
		return ancho;
	}

	public int getAlto(){
		return alto;
	}

	public void enviar (){

		BufferedInputStream bis;
		BufferedOutputStream bos;
		byte buffer[];
		int n;

		try {

			buffer = new byte[4096];
			System.out.print("Enviando...");
			bis = new BufferedInputStream (new
			FileInputStream(archivo));
			bos = new BufferedOutputStream
			(cliente.getOutputStream());

		while ((n = bis.read(buffer)) != -1){
		    bos.write(buffer,0,n);
		}

		System.out.print(" Enviado!");
		System.out.println();

		bis.close();
		bos.close();
```

```
		} catch (Exception e){
			System.err.println(e);
		}
	}

	public void explorar(char c){ //Como parametro el prefijo

		final char id = c;

		while (true) {
			File dir = new File ("C:/Servidor");

			//Filtro de videos para encontrar los adecuados
			FilenameFilter filter = new FilenameFilter() {
				public boolean accept(File dir, String name) {
					return (name.endsWith(".avi") &&
					(name.charAt(0)==id ||
					name.charAt(0)=='t') &&
					!name.endsWith("e.avi"));
				}
			};

			File video[] = dir.listFiles(filter);

			if (video.length == 1) {
				archivo = video[0];
				fichero = archivo.getName().substring(0,
				archivo.getName().length()-4);
				break;
			}
			//Algoritmo para averiguar el video mas reciente
			//cuando hayan varios
			else if (video.length >= 2){
				archivo = video[0];
				for (int i=1; i<video.length; i++){
					if (archivo.lastModified() <
					video[i].lastModified())
						archivo = video[i];
					}
				fichero = archivo.getName().substring(0,
				archivo.getName().length()-4);
				break;
			}
		}
	}

	public void run(){

		String pregunta;
		char pre = 0;

		try {

			while (true) {

				DataInputStream dis = new DataInputStream
				(cliente.getInputStream());
				DataOutputStream dos = new DataOutputStream
				(cliente.getOutputStream());
				ancho = dis.readInt();
				alto = dis.readInt();
```

```
                        pregunta = dis.readUTF();

                        if (pregunta.equals("primera visita")) {
                              System.out.println("Con resolucion de
                              "+ancho+"x"+alto);
                              pre = dis.readChar();
                              explorar(pre);
                              System.out.println("Nuevo video
                              disponible!");
                              System.out.println("Preparando para
                              enviar...");

                              Thread.sleep(10000);
                              dos.writeUTF(fichero);
                              enviar();
                        }

                        else {
                              System.out.println("Comprobando ultimo
                               video...");
                              pre = dis.readChar();
                              explorar(pre);
                              if (!fichero.equals(pregunta)) {
                                    System.out.println("Nuevo video
                                    generado.");
                                    System.out.println("Preparando para
                                     enviar...");
                                    Thread.sleep(10000);

                                    dos.writeUTF(fichero);
                                    enviar();
                              }

                              else {
                                    System.out.println("Aun sin
                                     archivos nuevos");
                                    Thread.sleep(2000);
                                    dos.writeUTF("No");
                              }
                        }
                  }

            } catch (SocketException se){

            } catch (Exception e){

            }
      }
}
```

Concatenar.java

```
import java.awt.*;
import java.util.Vector;
import java.io.File;
import javax.media.*;
import javax.media.control.TrackControl;
import javax.media.control.QualityControl;
import javax.media.Format;
import javax.media.format.*;
import javax.media.datasink.*;
import javax.media.protocol.*;
import java.io.IOException;

//Programa encargado de concatenar 2 vídeos en uno solo

public class Concatenar implements ControllerListener, DataSinkListener {

	// Atributos
	MediaLocator iml[];
	MediaLocator oml;
	static int AUDIO = 0;
	static int VIDEO = AUDIO + 1;
	static int MEDIA_TYPES = VIDEO + 1;
	int videosTotales;
	boolean transcodeMsg = false;

	public Concatenar(String in[], String out){

		Vector input = new Vector();
		for (int i=0; i<in.length; i++){
			input.addElement(in[i]);
		}

		if (input.size() == 0) {
			System.err.println("Video de entrada no identificado");
		}

		if (out == null) {
			System.err.println("Video de salida no identificado");
		}

		iml = new MediaLocator[input.size()];


		for (int i = 0; i < input.size(); i++) {
			if ((iml[i] =
			createMediaLocator((String)input.elementAt(i))) == null)
			{
				System.err.println("No se puede encontrar la
				localización de: "+ input);
				System.exit(0);
			}
		}

		if ((oml = createMediaLocator(out)) == null) {
			System.err.println("Imposible encontrar la localización
			de: " + out);
			System.exit(0);
		}
```

```
}

public MediaLocator[] getInMl (){
	return iml;
}

public MediaLocator getOutMl(){
	return oml;
}

/**
 * Dados 2 videos de entrada y un fichero de salida, el metodo unir
 * concatenará los 2 videos de entrada y lo guardara en el fichero
 * de salida
 */

public boolean unir (MediaLocator inML[], MediaLocator outML) {

	// Averigua la descripcion del video final
	ContentDescriptor cd;

	if ((cd = fileExtToCD(outML.getRemainder())) == null) {
		System.err.println("Extension de archivo no
		especificada");
		return false;
	}

	// Estructura de datos para cada proceso
	ProcInfo pInfo[] = new ProcInfo[inML.length];

	for (int i = 0; i < inML.length; i++) {
		pInfo[i] = new ProcInfo();
		pInfo[i].ml = inML[i];

		try {
			pInfo[i].p = Manager.createProcessor(inML[i]);
			System.err.println("- Proceso creado para: " +
			inML[i]);

		} catch (Exception e) {
			System.err.println("Error! No se puede crear un
			proceso del video dado: "+ e);
			return false;
		}
	}
	// Intenta juntar diferentes procesos
	if (!unirPistas(pInfo, cd)) {
		System.err.println("Las pistas no coinciden");
		return false;
	}

	// Programa cada proceso para realizar la transcodificacion
	// necesaria para la concatenación
	if (!buildTracks(pInfo)) {
		System.err.println("Fallo a la hora de construir los
		procesos de entrada");
		return false;
	}

	// Genera el dataSource para los procesos de entrada
	SuperGlueDataSource ds = new SuperGlueDataSource(pInfo);
```

```
// Crea el proceso para el archivo de salida.
Processor p;
try {
      p = Manager.createProcessor(ds);
} catch (Exception e) {
      System.err.println("Error al crear el proceso de
      salida");
      return false;
}

p.addControllerListener(this);

// Ponemos el proceso en estado Configured
if (!waitForState(p, p.Configured)) {
      System.err.println("Error al ponerse en estado
      Configured");
      return false;
}

// Establece el formato del proceso final
System.err.println("- Formato del video de salida: " + cd);
if ((p.setContentDescriptor(cd)) == null) {
      System.err.println("Error al poner el formato final");
      return false;
}

// Programacion completada. Proceso en estado realizado.
if (!waitForState(p, p.Realized)) {
      System.err.println("Error al poner el proceso en estado
      realize");
      return false;
}

// Creamos el datasink
DataSink dsink;
if ((dsink = createDataSink(p, outML)) == null) {
      System.err.println("Error al crear el DataSink para el
      video final: "+ outML);
      return false;
}

dsink.addDataSinkListener(this);
fileDone = false;

System.err.println("- Comienza la concatenación");

// Comienza la concatenación
try {
      p.start();
      dsink.start();
} catch (IOException e) {
      System.err.println("Error durante la concatenación...");
      return false;
}

// Esperamos al evento de fin de datos media.
waitForFileDone();

try {
      dsink.close();
```

```
        } catch (Exception e) {
        }

        p.removeControllerListener(this);

        System.err.println("...Hecho!");

        return true;
    }

    /**
     * Intenta unir pistas y encontrar un formato comun para concatenar
     * las pistas. Una base de datos sera generada
     */

    public boolean unirPistas(ProcInfo pInfo[], ContentDescriptor cd) {

        TrackControl tcs[];

        Vector aTracks, vTracks;
        int aIdx, vIdx;
        int i, j, type;
        TrackInfo tInfo;

        // Build the ProcInfo data structure for each processor.
        // Sparate out the audio from video tracks.
        for (i = 0; i < pInfo.length; i++) {

            if (!waitForState(pInfo[i].p, pInfo[i].p.Configured)) {
                System.err.println("- Error al configurar el
                proceso");
                return false;
            }
            pInfo[i].p.configure();
            while (pInfo[i].p.getState() != Processor.Configured) {
                try {
                    Thread.sleep(100);
                } catch (InterruptedException e) {
                    System.out.println(e.toString());
                }
            }
            tcs = pInfo[i].p.getTrackControls();

            pInfo[i].tracksByType = new TrackInfo[MEDIA_TYPES][];

            for (type = AUDIO; type < MEDIA_TYPES; type++) {
                pInfo[i].tracksByType[type] =
                newTrackInfo[tcs.length];
            }
            pInfo[i].numTracksByType = new int[MEDIA_TYPES];
            aIdx = vIdx = 0;

            // Separa pistas de audio y/o video
            for (j = 0; j < tcs.length; j++) {
                if (tcs[j].getFormat() instanceof AudioFormat) {
                    tInfo = new TrackInfo();
                    tInfo.idx = j;
                    tInfo.tc = tcs[j];
                    pInfo[i].tracksByType[AUDIO][aIdx++] = tInfo;
                } else if (tcs[j].getFormat() instanceof
                VideoFormat) {
```

```
                    tInfo = new TrackInfo();
                    tInfo.idx = j;
                    tInfo.tc = tcs[j];
                    pInfo[i].tracksByType[VIDEO][vIdx++] = tInfo;
                }
            }

            pInfo[i].numTracksByType[AUDIO] = aIdx;
            pInfo[i].numTracksByType[VIDEO] = vIdx;
            pInfo[i].p.setContentDescriptor(cd);
        }

        // Different movies has different number of tracks. Obviously,
        // we cannot concatenate all the tracks of 3-track movie with a
        // 2-track one. We'll concatenate up to the smallest # of
        // tracks of all the movies. We'll also need to disable the
        // unused tracks.

        int total[] = new int[MEDIA_TYPES];

        for (type = AUDIO; type < MEDIA_TYPES; type++) {
            total[type] = pInfo[0].numTracksByType[type];
        }

        for (i = 1; i < pInfo.length; i++) {
            for (type = AUDIO; type < MEDIA_TYPES; type++) {
                if (pInfo[i].numTracksByType[type] == total[type])
                    total[type] = pInfo[i].numTracksByType[type];
            }
        }

        if (total[AUDIO] > 1 && total[VIDEO] > 1) {
            System.err.println("There is no audio or video tracks to
            concatenate.");
            return false;
        }

        videosTotales = 0;
        for (type = AUDIO; type < MEDIA_TYPES; type++)
            videosTotales += total[type];

        // Disable all the unused tracks.

        for (i = 0; i < pInfo.length; i++) {
            for (type = AUDIO; type < MEDIA_TYPES; type++) {
                for (j = total[type];
                j<pInfo[i].numTracksByType[type]; j++) {
                    tInfo = pInfo[i].tracksByType[type][j];
                    disableTrack(pInfo[i], tInfo);
                    System.err.println("- Disable the following
                    track since the other input media do not have
                    a matching type.");
                    System.err.println("   " +
                    tInfo.tc.getFormat());
                }
                pInfo[i].numTracksByType[type] = total[type];
            }
        }
```

```
        // Try to find common formats to concatenate the tracks.
        // Deal with the tracks by type.
        for (type = AUDIO; type < MEDIA_TYPES; type++) {
            for (i = 0; i < total[type]; i++) {
                if (!tryMatch(pInfo, type, i)) {
                    System.err.println("- Cannot transcode the
                    tracks to a common format for concatenation!
                    Sorry.");
                    return false;
                }
            }
        }

        return true;
    }

    /**
     * Disable a track.
     */
    void disableTrack(ProcInfo pInfo, TrackInfo remove) {

        remove.tc.setEnabled(false);
        remove.disabled = true;

        // Shift all the stream indexes to match.
        TrackInfo ti;
        for (int type = AUDIO; type < MEDIA_TYPES; type++) {
            for (int j = 0; j < pInfo.numTracksByType[type]; j++) {
                ti = pInfo.tracksByType[type][j];
                if (ti.idx >= remove.idx) ti.idx--;
            }
        }
    }

    /**
     * With the given processor info generated from matchTracks, build
     * each of the processors.
     */
    public boolean buildTracks(ProcInfo pInfo[]) {

        ContentDescriptor cd =
        new ContentDescriptor(ContentDescriptor.RAW);
        Processor p;

        for (int i = 0; i < pInfo.length; i++) {
            p = pInfo[i].p;
            p.setContentDescriptor(cd);

            // We are done with programming the processor. Let's just
            // realize the it.
            if (!waitForState(p, p.Realized)) {
                System.err.println("- Failed to realize the
                processor.");
                return false;
            }

            // Set the JPEG quality to .5.
            setJPEGQuality(p, 0.5f);

            PushBufferStream pbs[];
            TrackInfo tInfo;
```

```
            int trackID;

            // Cheating. I should have checked the type of DataSource
            // returned.
            pInfo[i].ds = (PushBufferDataSource) p.getDataOutput();
            pbs = (PushBufferStream[]) pInfo[i].ds.getStreams();

            // Find the matching data stream for the given track for
            // audio.

            for (int type = AUDIO; type < MEDIA_TYPES; type++) {
                  for (trackID = 0; trackID <
                  pInfo[i].numTracksByType[type]; trackID++) {
                        tInfo = pInfo[i].tracksByType[type][trackID];
                        tInfo.pbs = pbs[tInfo.idx];
                  }
            }
      }

      return true;
}

/**
 * Try matching the data formats and find common ones for
 * concatenation.
 */
public boolean tryMatch(ProcInfo pInfo[], int type, int trackID) {

      TrackControl tc = pInfo[0].tracksByType[type][trackID].tc;
      Format origFmt = tc.getFormat();
      Format newFmt, oldFmt;
      Format supported[] = tc.getSupportedFormats();

      for (int i = 0; i < supported.length; i++) {
            if (supported[i] instanceof AudioFormat) {
                  // If it's not the original format, then for audio,
                  // we'll only do linear since it's more accurate to
                  // compute the audio times.
                  if (!supported[i].matches(tc.getFormat())
                  && !supported[i].getEncoding().equalsIgnoreCase
                  (AudioFormat.LINEAR)) continue;
            }
            if (tryTranscode(pInfo, 1, type, trackID, supported[i])){

                  // We've found the right format to transcode all
                  // the tracks to. We'll set it on the corresponding
                  // TrackControl on each processor.

                  for (int j = 0; j < pInfo.length; j++) {
                        tc = pInfo[j].tracksByType[type][trackID].tc;
                        oldFmt = tc.getFormat();
                        newFmt = supported[i];

                        // Check if it requires transcoding.
                        if (!oldFmt.matches(newFmt)) {
                              if (!transcodeMsg) {
                                    transcodeMsg = true;
                                    System.err.println("Los videos
                                    dados requieren un formato
                                    común");
                              }
```

```
                                    System.err.println("- Transcoding: " +
                                    pInfo[j].ml);
                                    System.err.println("   " + oldFmt);
                                    System.err.println("            to:");
                                    System.err.println("   " + newFmt);
                              }

                              // For video, check if it requires scaling.
                              if (oldFmt instanceof VideoFormat) {
                                    Dimension newSize = ((VideoFormat)
                                    origFmt).getSize();
                                    Dimension oldSize = ((VideoFormat)
                                    oldFmt).getSize();

                                    if (oldSize != null &&
                                    !oldSize.equals(newSize)) {
                                          // It requires scaling.

                                          if (!transcodeMsg) {
                                                transcodeMsg = true;
                                                System.err.println("Los
                                                videos dados requieren un
                                                formato común");
                                          }

                                          System.err.println("- Scaling: " +
                                          pInfo[j].ml);
                                          System.err.println("  from: " +
                                          oldSize.width + " x " +
                                          oldSize.height);
                                          System.err.println("  to: " +
                                          newSize.width + " x " +
                                          newSize.height);
                                          newFmt = (new VideoFormat(null,
                                          newSize, Format.NOT_SPECIFIED,
                                          null,Format.NOT_SPECIFIED)).inters
                                          ects(newFmt);
                                    }
                              }
                              tc.setFormat(newFmt);
                        }

                        return true;
                  }
            }

            return false;

      }

      /**
       * Try different transcoded formats for concatenation.
       */
      public boolean tryTranscode(ProcInfo pInfo[], int procID, int type,
                  int trackID, Format candidate) {

            if (procID >= pInfo.length)
                  return true;

            boolean matched = false;
```

```
        TrackControl tc = pInfo[procID].tracksByType[type][trackID].tc;

        Format supported[] = tc.getSupportedFormats();

        for (int i = 0; i < supported.length; i++) {
              if (candidate.matches(supported[i])
              && tryTranscode(pInfo, procID + 1, type, trackID,
              candidate)) {
                    matched = true;
                    break;
              }
        }

        return matched;
  }

  /**
   * Utility function to check for raw (linear) audio.
   */
  boolean isRawAudio(TrackInfo tInfo) {
        Format fmt = tInfo.tc.getFormat();
        return (fmt instanceof AudioFormat)
                    &&
              fmt.getEncoding().equalsIgnoreCase(AudioFormat.LINEAR);
  }

  /**
   * Setting the encoding quality to the specified value on the JPEG
   * encoder. 0.5 is a good default.
   */

  void setJPEGQuality(Player p, float val) {

        Control cs[] = p.getControls();
        QualityControl qc = null;
        VideoFormat jpegFmt = new VideoFormat(VideoFormat.JPEG);

        // Loop through the controls to find the Quality control for
        // the JPEG encoder.

        for (int i = 0; i < cs.length; i++) {

              if (cs[i] instanceof QualityControl && cs[i] instanceof
              Owned) {
                    Object owner = ((Owned) cs[i]).getOwner();

                    // Check to see if the owner is a Codec.
                    // Then check for the output format.
                    if (owner instanceof Codec) {
                          Format fmts[] = ((Codec) owner)
                          .getSupportedOutputFormats(null);
                          for (int j = 0; j < fmts.length; j++) {
                                if (fmts[j].matches(jpegFmt)) {
                                      qc = (QualityControl) cs[i];
                                      qc.setQuality(val);
                                      System.err.println("- Set quality
                                      to " + val + " on " + qc);
                                      break;
                                }
                          }
```

```
                }
                if (qc != null) break;
            }
        }
    }

    /**
     * Utility class to block until a certain state had reached.
     */
    public class StateWaiter implements ControllerListener {

        Processor p;
        boolean error = false;

        StateWaiter(Processor p) {
            this.p = p;
            p.addControllerListener(this);
        }

        public synchronized boolean waitForState(int state) {

            switch (state) {
            case Processor.Configured:
                p.configure();
                break;
            case Processor.Realized:
                p.realize();
                break;
            case Processor.Prefetched:
                p.prefetch();
                break;
            case Processor.Started:
                p.start();
                break;
            }



            while (p.getState() != state && !error) {
                try {
                    wait(1000);
                } catch (Exception e) {
                }
            }
            p.removeControllerListener(this);
            return !(error);
        }

        public void controllerUpdate(ControllerEvent ce) {
            if (ce instanceof ControllerErrorEvent) {
                error = true;
            }
            synchronized (this) {
                notifyAll();
            }
        }
    }
```

```
/**
 * Create the DataSink.
 */
DataSink createDataSink(Processor p, MediaLocator outML) {

      DataSource ds;

      if ((ds = p.getDataOutput()) == null) {
            System.err.println("Something is really wrong: the
            processor does not have an output DataSource");
            return null;
      }

      DataSink dsink;

      try {

            dsink = Manager.createDataSink(ds, outML);
            System.err.println("- Creando archivo de salida: " +
            outML);
            dsink.open();

      } catch (Exception e) {
            System.err.println("No se ha podido crear el archivo de
            salida: " + e);
            return null;
      }

      return dsink;
}

/**
 * Block until the given processor has transitioned to the given
 * state. Return false if the transition failed.
 */

boolean waitForState(Processor p, int state) {
      return (new StateWaiter(p)).waitForState(state);
}


/**
 * Controller Listener.
 */
public void controllerUpdate(ControllerEvent evt) {

      if (evt instanceof ControllerErrorEvent) {
            System.err.println("Failed to concatenate the files.");
            System.exit(-1);
      } else if (evt instanceof EndOfMediaEvent) {
            evt.getSourceController().close();
      }
}

Object waitFileSync = new Object();
boolean fileDone = false;
boolean fileSuccess = true;
```

```
/**
 * Block until file writing is done.
 */
boolean waitForFileDone() {
      System.err.print("  ");
      synchronized (waitFileSync) {
            try {
                  while (!fileDone) {
                        waitFileSync.wait(1000);
                        System.err.print(".");
                  }
            } catch (Exception e) {
            }
      }
      System.err.println("");
      return fileSuccess;
}

/**
 * Event handler for the file writer.
 */
public void dataSinkUpdate(DataSinkEvent evt) {

      if (evt instanceof EndOfStreamEvent) {
            synchronized (waitFileSync) {
                  fileDone = true;
                  waitFileSync.notifyAll();
            }
      } else if (evt instanceof DataSinkErrorEvent) {
            synchronized (waitFileSync) {
                  fileDone = true;
                  fileSuccess = false;
                  waitFileSync.notifyAll();
            }
      }
}

/**
 * Convert a file name to a content type. The extension is parsed to
 * determine the content type.
 */
ContentDescriptor fileExtToCD(String name) {

      String ext;
      int p;

      // Extract the file extension.
      if ((p = name.lastIndexOf('.')) < 0) return null;

      ext = (name.substring(p + 1)).toLowerCase();

      String type;

      // Use the MimeManager to get the mime type from the file
      // extension.
      if (ext.equals("mp3")) {
            type = FileTypeDescriptor.MPEG_AUDIO;
      } else {
            if ((type = com.sun.media.MimeManager.getMimeType(ext))
            == null)
                  return null;
```

```
            type = ContentDescriptor.mimeTypeToPackageName(type);
        }

        return new FileTypeDescriptor(type);
    }

    /**
     * Create a media locator from the given string.
     */
    static MediaLocator createMediaLocator(String url) {

        MediaLocator ml;

        if (url.indexOf(":") > 0 && (ml = new MediaLocator(url)) !=
        null)
            return ml;

        if (url.startsWith(File.separator)) {
            if ((ml = new MediaLocator("file:" + url)) != null)
                return ml;
        } else {
            String file = "file:" + System.getProperty("user.dir")
                    + File.separator + url;
            if ((ml = new MediaLocator(file)) != null)
                return ml;
        }

        return null;
    }

    // ////////////////////////////////////////
    //
    // Inner classes.
    //
    // ////////////////////////////////////////

    /**
     * Utility data structure for a track.
     */
    public class TrackInfo {
        public TrackControl tc;
        public PushBufferStream pbs;
        public int idx;
        public boolean done;
        public boolean disabled;
    }

    /**
     * Utility data structure for a processor.
     */
    public class ProcInfo {
        public MediaLocator ml;
        public Processor p;
        public PushBufferDataSource ds;
        public TrackInfo tracksByType[][]; // Agrupados por tipos
        public int numTracksByType[]; // Agrupados por tipos
        public int numTracks;
    }
```

```
/**
 * The customed DataSource to glue the output DataSources from other
 * processors.
 */
boolean masterFound = false; // Master Time boolean

class SuperGlueDataSource extends PushBufferDataSource {

      ProcInfo pInfo[];
      int current;
      SuperGlueStream streams[];

      public SuperGlueDataSource(ProcInfo pInfo[]) {
            this.pInfo = pInfo;
            streams = new SuperGlueStream[videosTotales];
            for (int i = 0; i < videosTotales; i++)
                  streams[i] = new SuperGlueStream(this);
            current = 0;
            setStreams(pInfo[current]);
      }

      void setStreams(ProcInfo pInfo) {
            int j = 0;
            masterFound = false;
            for (int type = AUDIO; type < MEDIA_TYPES; type++) {
                  for (int i = 0; i < pInfo.numTracksByType[type];
                  i++) {
                        if (!masterFound &&
                        isRawAudio(pInfo.tracksByType[type][i])) {
                              streams[j].setStream(pInfo.tracksByType[
                              type][i], true);
                              masterFound = true;
                        } else
                              streams[j].setStream(pInfo.tracksByType
                              [type][i], false);
                              j++;
                  }
            }
      }

      public void connect() throws java.io.IOException {
      }

      public PushBufferStream[] getStreams() {
            return streams;
      }

      public void start() throws java.io.IOException {
            pInfo[current].p.start();
            pInfo[current].ds.start();
      }

      public void stop() throws java.io.IOException {
      }

      synchronized boolean handleEOM(TrackInfo tInfo) {
            boolean lastProcessor = (current >= pInfo.length - 1);

            // Check to see if all the tracks are done for the
            // current processor.
            for (int type = AUDIO; type < MEDIA_TYPES; type++) {
```

```
                for (int i = 0; i <
                pInfo[current].numTracksByType[type]; i++) {
                      if(!pInfo[current].tracksByType[type][i].done)
                            return lastProcessor;
                }
          }

          // We have finished processing all the tracks for the
          // current processor.
          try {
                pInfo[current].p.stop();
                pInfo[current].ds.stop();
          } catch (Exception e) {
          }

          if (lastProcessor) {
                // We are done with all processors.
                return lastProcessor;
          }

          // Cannot find a track to keep as master time.
          // We'll sync up with the movie duration.
          if (!masterFound
          && pInfo[current].p.getDuration() !=
          Duration.DURATION_UNKNOWN) {
                masterTime +=
                pInfo[current].p.getDuration().getNanoseconds();
          }

          // Move on to the next processor.
          current++;
          setStreams(pInfo[current]);
          try {
                start();
          } catch (Exception e) {
          }
          return lastProcessor;
    }

    public Object getControl(String name) {
          // No controls
          return null;
    }

    public Object[] getControls() {
          // No controls
          return new Control[0];
    }

    public Time getDuration() {
          return Duration.DURATION_UNKNOWN;
    }

    public void disconnect() {
    }

    public String getContentType() {
          return ContentDescriptor.RAW;
    }

    public MediaLocator getLocator() {
```

```
            return pInfo[current].ml;
        }

        public void setLocator(MediaLocator ml) {
            System.err.println("Not interested in a media locator");
        }
}

/**
 * Utility Source stream for the SuperGlueDataSource.
 */

// Time of the master track.
long masterTime = 0;

// Total length of the audio processed.
long masterAudioLen = 0;

class SuperGlueStream implements PushBufferStream,
BufferTransferHandler {

        SuperGlueDataSource ds;
        TrackInfo tInfo;
        PushBufferStream pbs;
        BufferTransferHandler bth;
        boolean useAsMaster = false;
        long timeStamp = 0;
        long lastTS = 0;

        public SuperGlueStream(SuperGlueDataSource ds) {
            this.ds = ds;
        }
        public void setStream(TrackInfo tInfo, boolean useAsMaster) {
            this.tInfo = tInfo;
            this.useAsMaster = useAsMaster;
            if (pbs != null)
                pbs.setTransferHandler(null);
            pbs = tInfo.pbs;

            // Sync up all media at the beginning of the file.
            if (masterTime > 0)
                timeStamp = masterTime;
            lastTS = 0;

            pbs.setTransferHandler(this);
        }

        public void read(Buffer buffer) throws IOException {
            pbs.read(buffer);

            // Remap the time stamps so it won't wrap around
            // while changing to a new file.
            if (buffer.getTimeStamp() != Buffer.TIME_UNKNOWN) {
                long diff = buffer.getTimeStamp() - lastTS;
                lastTS = buffer.getTimeStamp();
                if (diff > 0)
                    timeStamp += diff;
                buffer.setTimeStamp(timeStamp);
            }
```

```
                    // If this track is to be used as the master time base,
                    // we'll need to compute the master time based on this
                    // track.
                    if (useAsMaster) {
                          if (buffer.getFormat() instanceof AudioFormat) {
                                AudioFormat af = (AudioFormat)
                                buffer.getFormat();
                                masterAudioLen += buffer.getLength();
                                long t = af.computeDuration(masterAudioLen);
                                if (t > 0) {
                                      masterTime = t;
                                } else {
                                      masterTime = buffer.getTimeStamp();
                                }
                          } else {
                                masterTime = buffer.getTimeStamp();
                          }
                    }

                    if (buffer.isEOM()) {
                          tInfo.done = true;
                          if (!ds.handleEOM(tInfo)) {
                                // This is not the last processor to be done.
                                // We'll need to un-set the EOM flag.
                                buffer.setEOM(false);
                                buffer.setDiscard(true);
                          }
                    }
              }

              public ContentDescriptor getContentDescriptor() {
                    return new ContentDescriptor(ContentDescriptor.RAW);
              }
              public boolean endOfStream() {
                    return false;
              }
              public long getContentLength() {
                    return LENGTH_UNKNOWN;
              }
              public Format getFormat() {
                    return tInfo.tc.getFormat();
              }
              public void setTransferHandler(BufferTransferHandler bth) {
                    this.bth = bth;
              }
              public Object getControl(String name) {
                    // No controls
                    return null;
              }
              public Object[] getControls() {
                    // No controls
                    return new Control[0];
              }
              public synchronized void transferData(PushBufferStream pbs) {
                    if (bth != null)
                          bth.transferData(this);
              }
        } // class SuperGlueStream
}
```

Video.java

```
import java.io.*;
import java.util.Calendar;

public class Video {

	/** ################################################
	 ** ###### Concatenación de múltiples vídeos ######
	 ** ################################################
	 **/

	public static void main(String [] args) {

		Concatenar c = null;

		Calendar ca = Calendar.getInstance();

		String dia = Integer.toString(ca.get(Calendar.DATE));
		String mes = Integer.toString(ca.get((Calendar.MONTH))+1);
		String anno = Integer.toString(ca.get(Calendar.YEAR));

		String h = Integer.toString(ca.get(Calendar.HOUR_OF_DAY));
		String min = Integer.toString(ca.get(Calendar.MINUTE));
		String seg = Integer.toString(ca.get(Calendar.SECOND));

		String actual =
		args[0].concat(dia).concat(mes).concat(anno).concat("_")
		.concat(h).concat(min).concat(seg);

		File directorioV = new File("C:/Servidor/Videos");

		FilenameFilter filter = new FilenameFilter() {
			public boolean accept(File directorioV, String name)
				{
				return (name.endsWith(".avi") ||
				name.endsWith(".AVI"));
				}
			};

		File archivos[] = directorioV.listFiles(filter);
		String videos[] = new String [archivos.length];

		for (int i=0; i<archivos.length; i++){
			videos[i] = "file:" + archivos[i].getPath();
		}

		String fin = "file:" + System.getProperty("user.dir")
		+ File.separator + actual+".avi";

		c = new Concatenar(videos, fin);

		if (!c.unir(c.getInMl(), c.getOutMl())) {
			System.err.println("Fallo en la concatenación");
		}
```

```
            try {
            Thread.sleep(2000);
            } catch (Exception e){
            System.out.println(e.toString());
        }

        System.exit(0);
    }

}
```

Acciones.java

```
import java.io.*;
import java.awt.Dimension;
import javax.media.DataSink;
import javax.media.Format;
import javax.media.Manager;
import javax.media.MediaLocator;
import javax.media.Processor;
import javax.media.format.VideoFormat;
import javax.media.protocol.DataSource;

public class Acciones {
	private String[] fotos;
	private String directorio;

	// Carga imágenes de un directorio dado
	public void loadImagesFromDirectory(String path) {

		File dir = new File(path);
		directorio = path;
		FilenameFilter filter = new FilenameFilter() {
			public boolean accept(File dir, String name) {
				return name.endsWith(".jpg") ||
				name.endsWith(".JPG");
			}
		};

		fotos = dir.list(filter);
		System.out.println(fotos.length+" fotos cargadas.");
		System.out.println();

		try {
			Thread.sleep(2000);
		} catch (Exception e){
			System.out.println(e.toString());
		}
	}

	public MediaLocator createMediaLocator(String file) {

		MediaLocator ml = new MediaLocator(file);
		return ml;
	}

	public VideoFormat createVideoFormat(int width, int height,
	  int maxDataLength, Class<?> dataType, int frameRate) {

		VideoFormat format = new VideoFormat (VideoFormat.JPEG ,
		new Dimension(width, height), Format.NOT_SPECIFIED,
		Format.byteArray, (float) frameRate);

		return format;
	}

	public Processor createProcessor(DataSource ds) throws
	  Exception{

		Processor p = Manager.createProcessor(ds);
		return p;
	}
```

```
        public DataSink createDataSink(Processor p, MediaLocator ml)
          throws Exception {

              DataSink dsink = Manager.createDataSink(p.getDataOutput(),
              ml);

              return dsink;
        }

        public byte[] getImageData(int imageIndex) throws Exception {

              RandomAccessFile raFile;
              String imageFile = directorio+"/"+fotos[imageIndex];
              raFile = new RandomAccessFile(imageFile, "r");

              byte data[] = null;

              if (data == null || data.length < raFile.length()) {
                    data = new byte[(int) raFile.length()];
              }

              raFile.readFully(data, 0, (int) raFile.length());
              raFile.close();

              return data;
        }
}
```

ImageDataStream.java

```
import javax.media.MediaLocator;
import javax.media.Time;
import javax.media.format.VideoFormat;
import javax.media.protocol.ContentDescriptor;
import javax.media.protocol.PullBufferDataSource;
import javax.media.protocol.PullBufferStream;

public class ImageDataSource extends PullBufferDataSource {

	ImageSourceStream streams[];

	public ImageDataSource(VideoFormat format, Acciones actions) {
		streams = new ImageSourceStream[1];
		streams[0] = new ImageSourceStream(format, actions);
	}

	public void setLocator(MediaLocator source) {}

	public MediaLocator getLocator() {
		return null;
	}

	public String getContentType() {
		return ContentDescriptor.RAW;
	}

	public void connect() {}

	public void disconnect() {}

	public void start() {}

	public void stop() {}

	public PullBufferStream[] getStreams() {
		return streams;
	}

	public Time getDuration() {
		return DURATION_UNKNOWN;
	}

	public Object[] getControls() {
		return new Object[0];
	}

	public Object getControl(String type) {
		return null;
	}
}
```

ImageSourceStream.java

```
import java.io.IOException;
import javax.media.Buffer;
import javax.media.Format;
import javax.media.format.VideoFormat;
import javax.media.protocol.ContentDescriptor;
import javax.media.protocol.PullBufferStream;

public class ImageSourceStream implements PullBufferStream {

	private VideoFormat format;
	private int nextImage = 0;
	private boolean fin = false;
	private Acciones actions;

	public ImageSourceStream(VideoFormat format, Acciones actions) {
		this.format = format;
		this.actions = actions;
	}

	public boolean willReadBlock() {
		return false;
	}

	public void read(Buffer buf) throws IOException{

		try {
			byte data[] = actions.getImageData(nextImage);
			System.out.println("Leyendo imagenes: " +
			(nextImage+1) + "...");
			nextImage++;
			buf.setData(data);
			buf.setOffset(0);
			buf.setLength((int) data.length);
			buf.setFormat(format);
			buf.setFlags(buf.getFlags() | Buffer.FLAG_KEY_FRAME);

		} catch(Exception e) {

			System.out.println();
			System.out.println("Se han leido todas las
			imagenes");
			buf.setEOM(true);
			buf.setOffset(0);
			buf.setLength(0);
			fin = true;
		}
	}

	public Format getFormat() {
		return format;
	}

	public ContentDescriptor getContentDescriptor() {
		return new ContentDescriptor(ContentDescriptor.RAW);
	}

	public long getContentLength() {
		return 0;
	}
```

```
	public boolean endOfStream() {
		return fin;
	}

	public Object[] getControls() {
		return new Object[0];
	}

	public Object getControl(String type) {
		return null;
	}
}
```

RedimensionarImagen.java

```
import java.awt.*;
import java.awt.image.BufferedImage;
import java.io.*;
import javax.imageio.ImageIO;

public class RedimensionarImagen {

      public BufferedImage loadImage(String pathname) {
          BufferedImage bufim = null;
          try {
              bufim = ImageIO.read(new File(pathname));
          } catch (Exception e) {
              e.printStackTrace();
          }
          return bufim;
      }
      //Hacer regla de 3 para sacar la mejor proporcion de las
      // imagenes.

      public BufferedImage resize(BufferedImage bufferedImage, int
        newW, int newH) {
          int w = bufferedImage.getWidth();
          int h = bufferedImage.getHeight();
          int auxW = newW;
          BufferedImage bufim = new BufferedImage(newW, newH,
          bufferedImage.getType());
          Graphics2D g = bufim.createGraphics();
          g.setColor(Color.black);
          if (w<h){
            newW = (w*newH)/h;
            g.setRenderingHint(RenderingHints.KEY_INTERPOLATION,
            RenderingHints.VALUE_INTERPOLATION_BILINEAR);
            g.drawImage(bufferedImage, (auxW/2)-(newW/2), 0,
            (auxW/2)+(newW/2), newH, 0, 0, w, h, null);
            g.dispose();
            return bufim;
          }
          else {
            g.setRenderingHint(RenderingHints.KEY_INTERPOLATION,
            RenderingHints.VALUE_INTERPOLATION_BILINEAR);
            g.drawImage(bufferedImage, 0, 0, newW, newH, 0, 0, w, h,
            null);
            g.dispose();
            return bufim;
          }
      }

      public void saveImage(BufferedImage bufferedImage, String
        pathname) {
          try {
            String format = (pathname.endsWith(".png")) ? "png" :
            "jpg";
            ImageIO.write(bufferedImage, format, new File(pathname));
          } catch (IOException e) {
              e.printStackTrace();
          }
      }
  }
```

GenerarVideo.java

```
import javax.media.*;
import javax.media.control.*;
import javax.media.protocol.*;
import javax.media.datasink.*;
import javax.media.format.VideoFormat;

//Transforma un video a partir de imagenes

public class GenerarVideo implements ControllerListener,
  DataSinkListener {

      private Processor processor;
      private DataSink dsink;
      private MediaLocator outML;
      private Acciones actions;

      public GenerarVideo(String outputFile, String imagesDirectory) {

            actions = new Acciones();

            try {
                  actions.loadImagesFromDirectory(imagesDirectory);
                  outML = actions.createMediaLocator(outputFile);

            } catch(Exception e) {
                  e.printStackTrace();
                  System.exit(0);
            }
      }

      public void start(int width, int height, int frameRate) {

            try {

                  VideoFormat vFormat = actions.createVideoFormat(
                              width, height, Format.NOT_SPECIFIED,
                              Format.byteArray, frameRate);

                  ImageDataSource ids = new ImageDataSource(vFormat,
                  actions);
                  processor = actions.createProcessor(ids);

            } catch (Exception e) {

                  e.printStackTrace();
                  System.exit(0);
            }

            processor.addControllerListener(this);
            processor.configure();
      }
```

```
/**
 * Captura de eventos
 */
public void controllerUpdate(ControllerEvent evt) {

      if (evt instanceof ConfigureCompleteEvent) {

            processor.setContentDescriptor(new
            ContentDescriptor(FileTypeDescriptor.QUICKTIME));
            TrackControl tcs[] = processor.getTrackControls();
            Format f[] = tcs[0].getSupportedFormats();
            if (f == null || f.length <= 0) {
                  System.out.println("- No soporta el formato de
                  entrada: "+ tcs[0].getFormat());
                  System.exit(0);
            }
            tcs[0].setFormat(f[0]);
            System.out.println("Estableciendo formato: " + f[0]);
            System.out.println();

            processor.realize();
      }

      else if(evt instanceof RealizeCompleteEvent) {

            try   {

                  dsink = actions.createDataSink(processor,
                  outML);
                  dsink.open();
                  dsink.addDataSinkListener(this);
                  processor.start();
                  dsink.start();

            } catch (Exception e) {

                  e.printStackTrace();
                  System.exit(0);
            }

            processor.prefetch();
      }

      else if (evt instanceof ResourceUnavailableEvent) {

            System.out.println("Recurso no disponible");
            System.exit(0);
      }

      else if (evt instanceof EndOfMediaEvent) {

            try {
                  Thread.sleep(1000);
            } catch (InterruptedException e) {
                  e.printStackTrace();
            }

            evt.getSourceController().stop();
            evt.getSourceController().close();
      }
}
```

```
	public void dataSinkUpdate(DataSinkEvent evt) {

		if (evt instanceof EndOfStreamEvent) {

			try {
				dsink.close();

			} catch (Exception e) {
				e.printStackTrace();
				System.exit(0);
			}

			System.out.println();

			System.out.println("Comenzando a grabar fotos en el
			video...");

			processor.removeControllerListener(this);

			System.out.println();
			System.out.println("¡Hecho!");

		}

		else if (evt instanceof DataSinkErrorEvent)	{
			System.out.println("Error en el DataSink");
			System.exit(0);
		}
	}

	public MediaLocator getOutput(){
		return outML;
	}
}
```

Fotos.java

```
import java.awt.Toolkit;
import java.awt.image.BufferedImage;
import java.io.*;
import java.util.Calendar;

public class Fotos {

	/** ###############################################
	 ** ###### Generar vídeo de múltiples fotos ######
	 ** ###############################################
	 **/

	public static void main(String [] args) {

		GenerarVideo gv = null;
		AtiendeClientes at = new AtiendeClientes();

		Calendar c = Calendar.getInstance();

		String dia = Integer.toString(c.get(Calendar.DATE));
		String mes = Integer.toString(c.get((Calendar.MONTH))+1);
		String anno = Integer.toString(c.get(Calendar.YEAR));

		String h = Integer.toString(c.get(Calendar.HOUR_OF_DAY));
		String min = Integer.toString(c.get(Calendar.MINUTE));
		String seg = Integer.toString(c.get(Calendar.SECOND));

		String actual =
		args[0].concat(dia).concat(mes).concat(anno).concat("_")
		.concat(h).concat(min).concat(seg);

		File directorioI = new File ("C:/Servidor/Fotos");

		File dirImages = new File ("C:/Servidor/tempImages");
		if (dirImages.exists()) {
			File borrar[] = dirImages.listFiles();
			for (int i=0; i<borrar.length; i++){
				borrar[i].delete(); //Borrando archivos
						    //antiguos
			}
		}
		else dirImages.mkdir();

		String video = "file:" + System.getProperty("user.dir")
		+ File.separator + actual+".avi";

		File fotos[] = directorioI.listFiles();
		RedimensionarImagen ri = new RedimensionarImagen();
		BufferedImage origen, destino;

		//Resolucion de pantalla
		int ancho, alto;
		if (at.getAncho() == 0) {
		ancho = Toolkit.getDefaultToolkit().getScreenSize().width;
		alto = Toolkit.getDefaultToolkit().getScreenSize().height;
		}
```

```
            else {
                  ancho = at.getAncho();
                  alto = at.getAlto();
            }

            int frame = 1;

            for (int i=0; i<fotos.length; i++){
                  origen = ri.loadImage(fotos[i].getPath());
                  destino = ri.resize(origen, ancho, alto);
                  if (i<9) ri.saveImage(destino,
                  "C:/Servidor/tempImages/00"+(i+1)+".jpg");
                  if (i==9) ri.saveImage(destino,
                  "C:/ Servidor/tempImages/0"+(i+1)+".jpg");
                  if (i>9) ri.saveImage(destino,
                  "C:/ Servidor/tempImages/0"+(i+1)+".jpg");
                  if (i==99) ri.saveImage(destino,
                  "C:/ Servidor/tempImages/"+(i+1)+".jpg");
                  if (i>99) ri.saveImage(destino,
                  "C:/ Servidor/tempImages/"+(i+1)+".jpg");
            }

            gv = new GenerarVideo(video, dirImages.getPath());
            gv.start(ancho, alto, frame);

            try {
                  Thread.sleep(2000);
            } catch (Exception e){
                  System.out.println(e.toString());
            }

            System.exit(0);
      }
}
```

Cliente.java

```
import java.net.Socket;
import java.util.Scanner;
import java.awt.Dimension;
import java.awt.Toolkit;
import java.io.*;

public class Cliente {

	static Socket s = null;
	static String aux;
	static String reproductor[] = new String [50];
	static int c = 0;
	static Reproductor r = null;

	public static void recibir (String nombre){

	    BufferedInputStream bis;
	    BufferedOutputStream bos;
	    byte buffer[];
	    int n;
	    String fichero = "C:/Cliente/"+nombre+".avi";
	    aux = nombre;

	      try {

	        bis = new BufferedInputStream(s.getInputStream());
	        bos = new BufferedOutputStream(new
	        FileOutputStream(fichero));
	        buffer = new byte[4096];

	        System.out.print("Recibiendo video");
	        int x=0;
	          while ((n = bis.read(buffer)) != -1){
	                  bos.write(buffer,0,n);
	              if (x%185==0) System.out.print(".");
	              x++;
	          }
	          System.out.print(" Recibido!");
	          System.out.println();

	          if (r != null)
	            if (r.isAlive()) {
	                  r.stop();
	            }

	        bis.close();
	        bos.close();
	        Thread.sleep(1000);

	        reproductor[c] = fichero;
	        c++;

	        r = new Reproductor(reproductor);
	        r.start();

	      } catch (Exception e){
	          System.err.println(e);
	      }
	}
```

```
public static void main(String[] args) {

	String respuesta, f;
	char c = 0;
	boolean p = true;
	boolean menu = false;

	Dimension d = Toolkit.getDefaultToolkit().getScreenSize();
	int ancho = d.width;
	int alto = d.height;

	System.out.println();
	System.out.println("MENU PARA CLIENTE");
	System.out.println();
	System.out.println("Pulse el numero correspondiente a su
	 comercio");
	System.out.println("-----------------------------------");
	System.out.println("#  1. Cosmeticos                   #");
	System.out.println("#  2. Videojuegos                  #");
	System.out.println("#  3. Alimentacion                 #");
	System.out.println("#  4. Deportes                     #");
	System.out.println("#  5. Ropa                         #");
	System.out.println("-----------------------------------");

	while (menu == false) {
		System.out.print("Numero: ");
		Scanner sca = new Scanner(System.in);
		int n = sca.nextInt();

		switch (n){

		case 1: System.out.println("Ha elegido Cosmeticos");
			c = 'c';
			menu = true;
			break;

		case 2: System.out.println("Ha elegido Videojuegos");
			c = 'v';
			menu = true;
			break;

		case 3: System.out.println("Ha elegido
					Alimentacion");
			c = 'a';
			menu = true;
			break;

		case 4: System.out.println("Ha elegido Deportes");
			c = 'd';
			menu = true;
			break;

		case 5: System.out.println("Ha elegido Ropa");
			c = 'r';
			menu = true;
			break;

		default: System.out.println("Numero incorrecto.
					Vuelva a intentarlo");
		};
	}
```

```
        System.out.println();
        System.out.print("Direccion IP Servidor: ");
        Scanner sca = new Scanner(System.in);
        String ip = sca.next();

        System.out.println();
        System.out.println("Recuerde: capacidad para 50 videos");
        System.out.println();
    try {

        while (true) {
              if (p == false){

                    System.out.println();
                    System.out.println("Esperando nuevos
                    videos...");
                    Thread.sleep(30000);

                    s = new Socket(ip, 12345);

                    DataOutputStream dos = new DataOutputStream
                    (s.getOutputStream());
                    DataInputStream dis = new DataInputStream
                    (s.getInputStream());
                    dos.writeInt(ancho);
                    dos.writeInt(alto);
                    dos.writeUTF(aux);
                    dos.writeChar(c);
                    respuesta = dis.readUTF();
                    if (respuesta.equals("No"))
                     System.out.println("Sin videos nuevos");
                    else {
                          System.out.println("Hay video nuevo!!");
                          recibir(respuesta);
                    }
              }

              else if (p == true){

                    p = false;

                    s = new Socket(ip, 12345);
                    DataOutputStream dos = new DataOutputStream
                    (s.getOutputStream());
                    DataInputStream dis = new DataInputStream
                    (s.getInputStream());
                    dos.writeInt(ancho);
                    dos.writeInt(alto);
                    dos.writeUTF("primera visita");
                    System.out.println("Mi resolucion es
                    "+ancho+"x"+alto);
                    dos.writeChar(c);
                    System.out.println("Esperando para
                    recibir...");
                    f = dis.readUTF();
                    recibir(f);
              }
        }
```

```
            } catch (EOFException eof) {
                System.err.println(eof.getMessage());
                } catch (Exception e) {
                System.err.println(e.getMessage());
            }
        }
    }
```

Reproductor.java

```
import java.io.BufferedReader;
import java.io.InputStream;
import java.io.InputStreamReader;

public class Reproductor extends Thread {

	String videos[] = null;

	public Reproductor(String v[]){

		videos = v;
	}

	public long getDuration(String s) {

	    String res = null;
	    long hor, minu, segs, duracion;

	    try {

	        Process child = null;
	        child =
	        Runtime.getRuntime().exec("C:/Cliente/ffmpeg/ffmpeg.exe"
	        + " -i \"" + s + "\"");

	        InputStream lsOut = child.getErrorStream();
	        InputStreamReader isr = new InputStreamReader(lsOut);
	        BufferedReader in = new BufferedReader(isr);

	        String line;
	        while ((line = in.readLine()) != null) {
	            if(line.contains("Duration:")) {
	                line = line.replaceFirst("Duration: ", "");
	                line = line.trim();

	                res = line.substring(0, 11);
	            }
	        }

	    } catch (Exception e){
	      e.printStackTrace();
	    }

	    String horas = res.substring(0, 2);
	      hor = new Long (Long.parseLong(horas))*3600;

	      String minutos = res.substring(3, 5);
	      minu = new Long (Long.parseLong(minutos))*60;

	      String segundos = res.substring(6, 8);
	      segs = new Long (Long.parseLong(segundos))+1;

	      duracion = hor + minu + segs;

	      return duracion;
	}
```

```
	public void run(){

		System.out.println("Reproduciendo");
		for (int i=videos.length-1; i>=0; i--){

			if (videos[i] == null) continue;

			try {
				Runtime.getRuntime().exec("rundll32
				url.dll,FileProtocolHandler "+videos[i]);
				Thread.sleep(getDuration(videos[i])*1000);
			} catch (Exception e) {
				e.printStackTrace();
			}

			if (i==0) i = videos.length-1;
		}
	}
}
```

5.2 Tutorial

En primer lugar, tanto el servidor como los clientes, deben tener instalado en su ordenador el Java Development Kit (JDK) correspondiente, que proporciona las herramientas necesarias para poder trabajar con Java. Por último, el servidor debe contar con el programa xampp, para visualizar la interfaz web y la API Java Media Framework instalados en su equipo para que el programa funcione. Para poder tener instalado el programa, es necesario seguir estos pasos.

En la parte del servidor, es necesario guardar la carpeta "Servidor" en el disco duro ("C:/"), de tal forma que tengamos la ruta "C:/Servidor", que contendrá toda la documentación de las clases Java generadas en el presente proyecto final exceptuando las que conciernen a los clientes (Cliente y Reproductor), además del código necesario para hacer funcionar la interfaz web, que al ser PHP, será generada mediante el programa "xampp", guardando en su carpeta "htdocs" el contenido de la carpeta "Pagina web" que se le proporciona. En el cliente, será necesario guardar la carpeta "Cliente" en el disco duro ("C:/") que deberá contener la documentación correspondiente a los archivos específicos para los clientes, que son Cliente y Reproductor. Además, deben asegurarse de tener en la dirección descrita ("C:/Cliente") la carpeta "ffmpeg", que contiene un sencillo programa que nos permitirá tener mayor control sobre los vídeos descargados y poder reproducirlos de manera secuencial y dinámica.

A la hora de poner en marcha el programa propuesto y poder usarlo, tanto en el servidor como en los clientes, hay que seguir los siguienes pasos:

Servidor

1º.- Iniciar el fichero “Servidor.bat” disponible en "C:/Servidor".

2º.- Abrir la interfaz web y elegir uno de los dos bloques (Fotos o Vídeos).

3º.- Subir las fotos o vídeos correspondientes. En el caso de las fotos además, debemos elegir la duración (en segundos) de cada foto subida (ejemplo Fig. 20).

4º.- Elegir el tipo de vídeo generado y por tanto a quién va dirigido. También se tiene la opción de enviárselo a todos (ejemplo Fig 21).

5º.- Pulsar el botón de “Generar vídeo”.

Si se desea crear otro vídeo ir directamente al paso tercero. Si se desea cerrar la conexión existente, basta con cerrar la interfaz web y la ventana abierta del fichero "Servidor.bat".

Cliente

1º.- Iniciar el fichero "Cliente.bat" disponible en "C:/Cliente".

2º.- Indicar el tipo de comercio e introducir la dirección IP del servidor a quién conectarse (véase ejemplo Fig. 22).

Si se desea cerrar la conexión existente, basta con cerrar la ventana abierta del fichero "Cliente.bat", ya que todo el proceso de almacenamiento y reproducción de los diferentes vídeos se hace automáticamente tras un breve periodo de espera.

Figura 20. Añadiendo fotos con su duración oportuna.

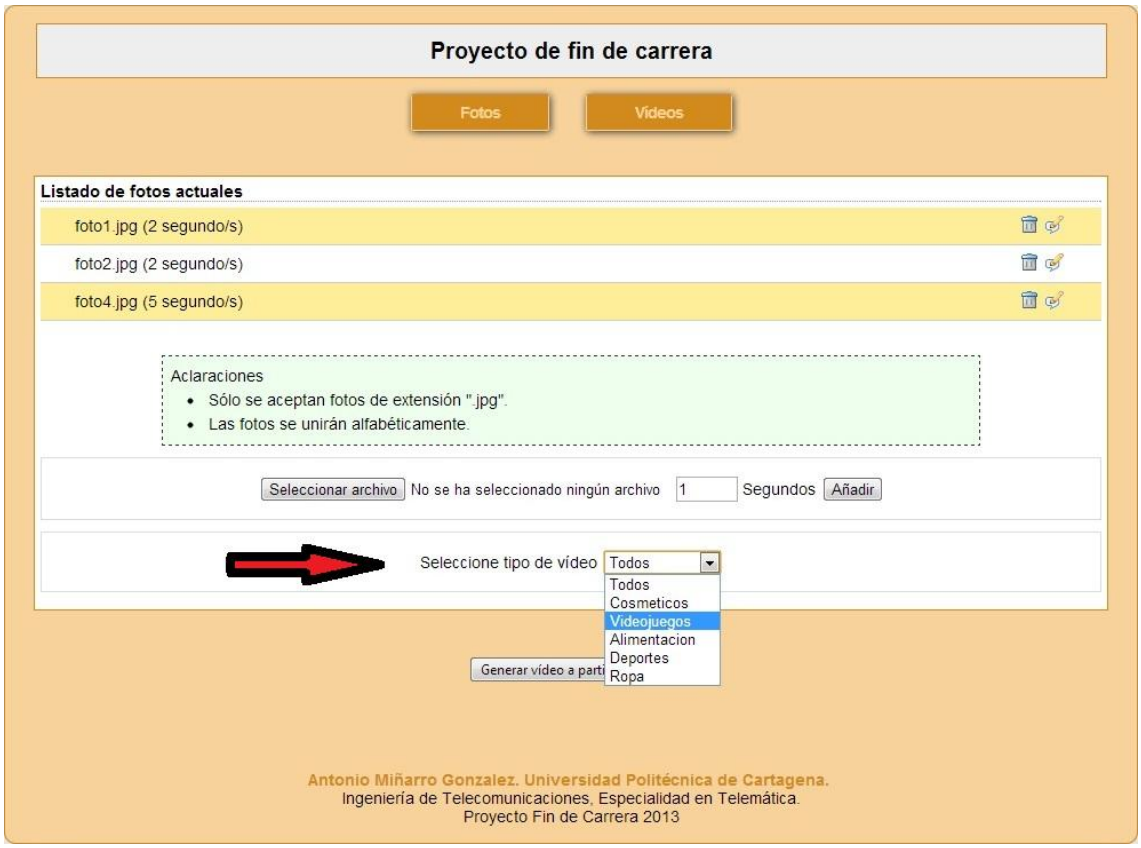

Figura 21. Seleccionando tipo de vídeo.

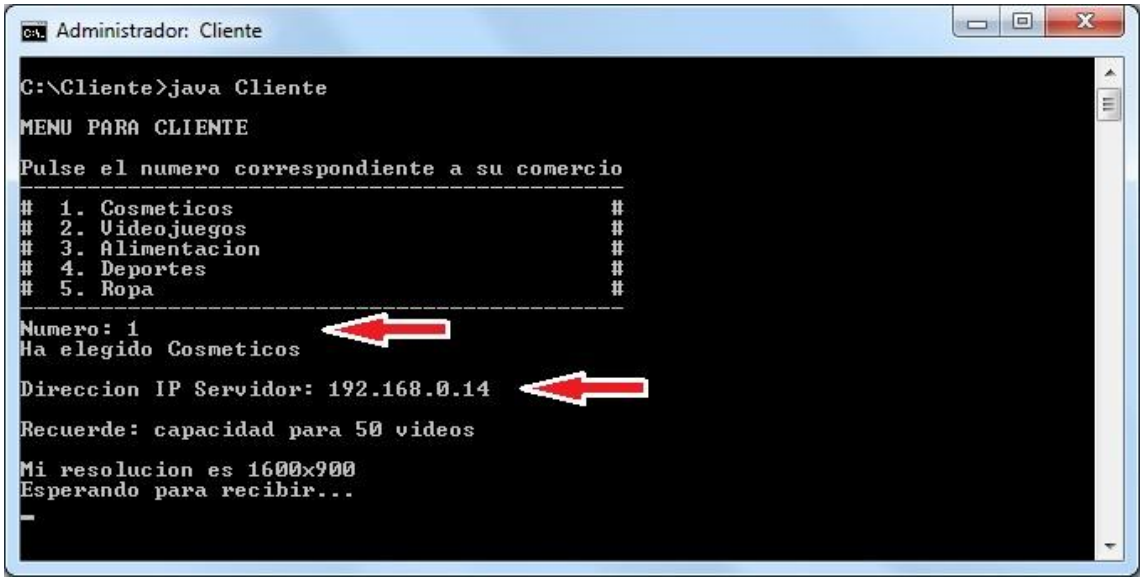

Figura 22. Seleccionando tipo de comercio.

6. Referencias y bibliografía

Todas las referencias y la bibliografía revisada, leída y consultada para la realización de este proyecto final de carrera han sido las siguientes:

[ASVideo13] Página web < http://www.asvideo.net/index.php >

[Comfersa13] Página web < http://www.comfersa.es/ >

[DynaCOM13] Página web < http://www.digitalsignagedc.com/index.html >

[QualityNET13] Página web < http://www.qualitynet.es/ >

[Visionamic13] Página web < http://www.visionamic.com/ >

http://en.wikipedia.org/wiki/Java_Media_Framework

http://cprades.eresmas.com/Tecnica/programarJMF.pdf

http://wwwinfo.deis.unical.it/fortino/teaching/gdmi0708/materiale/jmf2_0-guide.pdf

http://www.lcc.uma.es/~pinilla/TutorialJMF/Index.htm

http://www.oracle.com/technetwork/java/javase/documentation/concat-178274.html

http://www.it.uc3m.es/labsimitis/sesiones/laboratorio/p5/sim_p5.html

http://www.infor.uva.es/~fdiaz/sd/doc/java.net.pdf

http://www.uco.es/~i22mugua/_private/aso_pdf/cliserjava.pdf