UNIVERSIDAD DE LAS PALMAS DE GRAN CANARIA
Escuela de Ingeniería Informática

Trabajo Final de Grado en Ingeniería Informática

Gestión de Vídeo Distribuido

Bycor Sánchez Sánchez

Tutores
Antonio Carlos Domínguez Brito
Modesto Castrillón Santana

20 de junio de 2013 version 1.0

Agradecimientos

Antes de empezar con el presente documento quisiera mostrar mi agradecimiento a todas las personas que me han apoyado y animado mientras realizaba este proyecto.

A mis compañeros y amigos, quienes me han alentado a continuar pese a las adversidades que iban surgiendo por el camino.

También dar las gracias a mis tutores, muy especialmente a Antonio Carlos Domínguez Brito quien siempre ha estado ahí para aconsejarme y ayudarme en todo lo que fuera necesario.

Me gustaría también nombrar a mis padres y hermanos, quienes en todo momento me han brindado su apoyo incondicional.

Finalmente, a esa persona que día tras día me ha animado para que este proyecto llegara a ser una realidad, quien ha alimentado mi interés por seguir adelante y ha sido la fuente de mi inspiración. Un agradecimiento más que merecido para Delioma Artiles Segura.

A todos, muchas gracias.

Resumen

El principal objetivo de este Trabajo Final de Grado (TFG) fue la creación de un sistema de gestión de vídeo distribuido utilizando cámaras de videovigilancia IP. Esta propuesta surgió a partir de la idea de ofrecer un acceso simultáneo, tanto online como offline, a las secuencias de vídeo generadas por una red de cámaras IP en un entorno dado.

El resultado obtenido fue una infraestructura software ampliable que ofrece al usuario una serie de funcionalidades con cámaras de red, abstrayéndolo de detalles internos. El trabajo está compuesto por tres elementos claramente diferenciados: integración de cámaras IP, almacenamiento en vídeo y creación del sistema de vídeo distribuido.

La integración de cámaras IP tiene como objetivo comunicar al equipo con la cámara de red para la obtención del flujo de imágenes que transmite. Dicha comunicación se establece vía HTTP[14] (Hypertext Transfer Protocol) gracias a la interfaz de programación (API) de la que disponen estos dispositivos.

El segundo elemento, el almacenamiento en vídeo, tiene como función guardar las imágenes de la cámara IP en archivos de vídeo. De esta manera se ofrece su posterior visualización en diferido.

Finalmente, el sistema de vídeo distribuido permite la reproducción simultánea de múltiples vídeos grabados por la red de cámaras IP. Adicionalmente, vídeos grabados por otros dispositivos también son admitidos.

El material desarrollado dispone del potencial necesario para convertirse en una herramienta libre de amplio uso en sistemas UNIX para cámaras IP, así como suponer la base de futuros proyectos relacionados con estos dispositivos.

Abstract

The main objective of this Trabajo Final de Grado (TFG) was the creation of a distributed video management system using surveillance IP cameras. This proposal arises from the idea of offering simultaneous access, online and offline, to video sequences generated by an IP camera network in a giving environment.

As a result, we obtained an upgradeable software infrastructure which provides the user with several functionalities with network cameras, abstracting him from internal details. The work can be clearly divided into three different parts: IP camera integration, video storage and distributed video system.

The aim of IP camera integration is to allow the communication between the computer and a network camera to obtain its image stream. That communication is established via HTTP[14] (Hypertext Transfer Protocol) thanks to the Application Programming Interface (API) of these devices.

The video storage main function is to save the IP camera image streams into video files. In this way recorded video sequences can be watched anytime.

Finally, the distributed video system allows the user to play multiple videos recorded simultaneously by the IP camera network. Additionally, videos recorded by other devices are accepted as well.

The material developed has the potential to be a free tool widely used on UNIX systems for IP cameras, as well as become the base of future projects associated with these devices.

Índice General

Capítulo 1

Introducción

1.1. Estado Actual

Nos encontramos en una época en la cual los sistemas de videovigilancia son muy utilizados, especialmente en comercios o zonas públicas. Dichos sistemas permiten la monitorización de numerosos espacios, tanto públicos como privados, con el fin de controlar fundamentalmente las actividades que en ellos se desarrollan. De este modo se puede llegar a garantizar la seguridad de bienes y personas, monitorizar un entorno dado y controlar presencias, entre otras posibilidades.

Históricamente los sistemas de videovigilancia eran conformados por cámaras de vigilancia tanto analógicas como digitales. No obstante, hace menos de dos décadas surgieron un nuevo tipo de cámaras denominadas **cámaras de red** o **cámaras IP**, las cuales permiten la emisión de las imágenes directamente por la red (intranet o internet) sin necesidad de un ordenador "pasarela". Se pueden ver algunos tipos de estas cámaras en la figura 1.1.

Figura 1.1: Diferentes modelos de cámaras IP pertenecientes a la compañía Axis Communications.

Concretamente la primera cámara IP fue creada por la empresa Axis Communications en 1996, la cual contenía un servidor web interno. Con el paso de los años (a partir de 1999) estas cámaras ya utilizan internamente una versión de Linux para operar.

Entre las principales características de las cámaras IP se encuentran:

- Compresión y envío de imágenes sobre la red.
- Requiere un mínimo de ancho de banda en la red para evitar latencias y conseguir una reproducción fluida.
- Soporte bajo conectividad inalámbrica (muchos modelos), lo que otorga **flexibilidad** a la hora de posicionarlas en sitios concretos. La conectividad inalámbrica está sujeta a que la señal llegue al dispositivo.
- Fácilmente **accesible** desde cualquier lugar con acceso a internet una vez configurada.
- Permite métodos de **autentificación** para restringir accesos.
- Puede encriptar las imágenes antes de enviarlas, aunque ello conlleva una considerable latencia.
- Capacidad para hacer zoom, enfocar o mover la cámara. Disponible sólo en algunos tipos.
- Inclusión de sensores para detectar cercanía, movimiento, etc. Dependiendo del modelo.

La instalación de estas cámaras no es trivial. Su conexión se realiza directamente a la red con la utilización de un cable ethernet o de manera inalámbrica, habiendo sido configurada previamente. En la figura 1.2 se puede ver un ejemplo de una instalación con cámaras IP con diferentes tipos de conexiones.

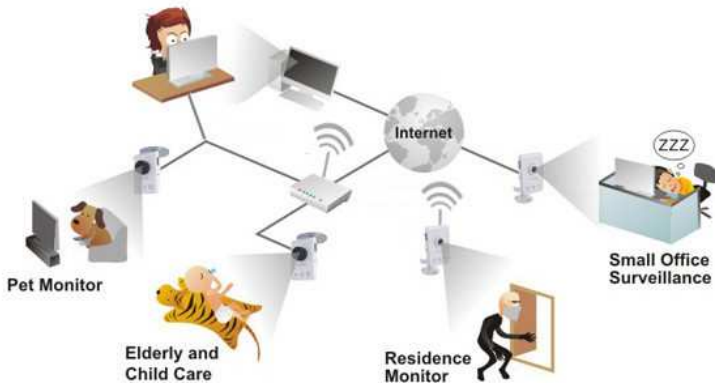

Figura 1.2: Esquema de una instalación tipo de cámaras ip.

En este Trabajo Fin de Grado se aborda la creación de un sistema que sea capaz de gestionar una red conformada por este tipo de cámaras.

Entre las funcionalidades que se desean implementar se encuentra el acceso a estos dispositivos de red para la visión en tiempo real de sus imágenes, posibilitar la grabación de las secuencias de vídeo para su visualización en diferido y generar un sistema de vídeo que ofrezca la capacidad de reproducir de manera integrada los múltiples archivos de vídeo grabados por las cámaras IP que conforman la red de videovigilancia.

1.2. Estructuración del documento

Este documento se organiza en 13 capítulos:

- **Introducción**: este capítulo.
- **Objetivos**: en este capítulo se realiza una exposición general de los hitos que se pretenden alcanzar en cada una de las etapas en las que está dividido el TFG.
- **Competencias**: se exponen las competencias cubiertas con este trabajo, indicando los apartados en las que éstas se ven satisfechas.
- **Pliego de condiciones**: en este apartado se analiza el pliego de condiciones requerido por una de las competencias expuestas en el capítulo anterior.
- **Aportaciones**: análisis de las contribuciones que el TFG aporta a nuestro entorno social, científico y técnico.
- **Normativa y legislación**: se manifiestan las normativas y legislaciones que afectan al presente trabajo.
- **Requisitos**: se especifican los requisitos tantos hardware como software asociados al proyecto.
- **Metodología y plan de trabajo**: en este capítulo se expone la metodología seguida a la hora de desarrollar el TFG y la planificación llevada a cabo para cumplir los objetivos planteados inicialmente.
- **Análisis del problema**: se describen en detalle la serie de recursos y herramientas utilizadas, explicando su funcionamiento, ventajas que presentan y argumentando su elección.
- **Diseño e implementación**: todos los detalles relativos al diseño, desarrollo e implementación de la infraestructura para cámaras IP, grabación de vídeo y sistema de gestión de vídeo distribuido se encuentran en este capítulo.
- **Pruebas**: demostraciones realizadas para certificar la correctitud del código generado en las diferentes etapas de desarrollo.
- **Conclusiones**: se presentan las conclusiones extraídas del desarrollo del proyecto. Además se plantean posibles líneas de desarrollo futuro a partir del trabajo realizado.

- **Manual de usuario**: explicación de los pasos a seguir por cualquier usuario que desee utilizar las aplicaciones creadas. En él se indica cómo realizar las instalaciones necesarias, tanto software como hardware, así como la obtención y ejecución del código desarrollado.

Capítulo 2

Objetivos

A continuación se especifican los diferentes objetivos a alcanzar con el presente Trabajo Fin de Grado.

2.1. Objetivos académicos

Desde este punto de vista académico, este TFG se ha orientado al aprovechamiento de los conocimientos adquiridos durante la titulación. De entre ellos se destacan las siguientes materias:

- Algoritmos, programación y estructuras de datos
- Sistemas empotrados y de tiempo real
- Algoritmos y programación paralela
- Sistemas operativos
- Interfaces humanas
- Redes

2.2. Objetivos del trabajo

La finalidad del trabajo es el desarrollo de un sistema de gestión de vídeo distribuido utilizando cámaras IP. La idea motora de esta propuesta es el acceso tanto online como offline a la grabación simultánea (paralela y/o concurrente) realizada por una red de cámaras IP distribuidas en un entorno dado.

De este modo se definen una serie de objetivos a alcanzar para el correcto desarrollo del sistema de gestión de vídeo. Dichos objetivos son los siguientes:

- **Integración de cámaras IP**. Desarrollo de una infraestructura software que integre cámaras IP abstrayendo elementos como: detalles propios del dispositivo, cómo se realiza la comunicación, cuál es el formato de los datos, entre otros. La principal función del software desarrollado será la obtención de imágenes de la cámara IP.

- **Almacenamiento en vídeo**. Desarrollo de infraestructura software que almacene las imágenes de cámaras IP en vídeo. De esta manera se logrará el acceso y gestión de forma offline a las imágenes de las cámaras de red. El software desarrollado hará uso de la integración de cámaras IP, objetivo anterior, para la generación de vídeo a partir de sus imágenes.

- **Sistema de vídeo distribuido**. Desarrollo de un sistema de gestión de vídeo distribuido que permita la gestión, reproducción y acceso integrado a múltiples secuencias de vídeo grabadas por una red de cámaras IP. Este sistema permitirá un acceso offline y simultáneo a las imágenes grabadas por múltiples cámaras IP.

Capítulo 3

Competencias

A continuación se relacionan explícitamente las diferentes competencias cubiertas en el presente trabajo, indicando en cada caso en qué parte del documento son satisfechas.

3.1. CII01

Capacidad para diseñar, desarrollar, seleccionar y evaluar aplicaciones y sistemas informáticos, asegurando su fiabilidad, seguridad y calidad, conforme a principios éticos y a la legislación y normativa vigente.

Esta competencia ha sido desarrollada a lo largo del documento en los capítulos de "Introducción", "Análisis del problema" y "Diseño e implementación"; secciones 1, 9 y 10, respectivamente. En dichos capítulos se muestran las causas y los hechos derivados de las decisiones de diseño tomadas.

3.2. CII02

Capacidad para planificar, concebir, desplegar y dirigir proyectos, servicios y sistemas informáticos en todos los ámbitos, liderando su puesta en marcha y su mejora continua y valorando su impacto económico y social.

En el capítulo "Metodología y planificación", sección 8, se indica el sistema de organización utilizado para alcanzar los objetivos propuestos. La valoración del impacto social y económico queda reflejada por su parte en el capítulo "Aportaciones", sección 5.

3.3. CII04

Capacidad para elaborar el pliego de condiciones técnicas de una instalación informática que cumpla los estándares y normativas vigentes.

En el capítulo de "Pliego de condiciones", sección 4, queda cubierta esta competencia especificando las condiciones asociadas a este Trabajo Final de Grado.

3.4. CII18

Conocimiento de la normativa y la regulación de la informática en los ámbitos nacional, europeo e internacional.

El cumplimiento de esta competencia queda descrito en el capítulo de "Normativa y Legislación", sección 6.

Capítulo 4

Pliego de Condiciones

4.1. Objeto de este pliego

El pliego de condiciones técnicas tiene por objeto la definición de condiciones generales, técnicas y económicas asociadas a la ejecución y utilización del software para la gestión de vídeo distribuido.

4.2. Pliego de condiciones generales

El presente trabajo está destinado al desarrollo de un sistema de gestión de vídeo distribuido para la monitorización de una red de videovigilancia constituida por cámaras IP. En líneas generales el sistema dispone de las siguientes características:

- Capacidad para la visión online, en tiempo real, de las imágenes transmitidas por una cámara IP del sistema de vigilancia.
- Almacenamiento en vídeo de las imágenes suministradas por las cámaras IP, de forma que se posibilite el acceso offline a las secuencias grabadas.
- Sistema de gestión que permite el acceso, reproducción y gestión de los vídeos grabados por la red de cámaras IP de manera integrada. Los múltiples vídeos serán reproducidos conjuntamente respetando su temporalidad, momento en que fueron grabados.

Cuando el uso del material desarrollado conlleve el tratamiento de datos personales se ha de aplicar la legislación vigente en España. En la actualidad la ley que regula este tema es la Ley Orgánica de Protección de Datos de Carácter Personal (LOPD)[38]. Además, en el ámbito de la vigilancia existe la Instrucción 1/2006, de 8 de noviembre[39] que trata de adecuar la LOPD al tratamiento de imágenes que contengan personas físicas identificadas o identificables.

4.3. Pliegos de especificaciones técnicas

El pliego de condiciones técnicas dispone de dos apartados diferenciados:

4.3.1. Especificaciones de materiales y equipos

Los materiales de los cuales se compone el trabajo son descritos a continuación. Es necesario diferenciar entre hardware y software:

- Hardware:
  - Cámaras IP de la compañía Axis Communications[3]. Estos dispositivos compondrán la red de vigilancia.
  - Redes TCP/IP fijas e inalámbricas para la conexión de cámaras de red.
  - Cableado ethernet. Necesario para la configuración inicial de las cámaras y para su conexión en redes fijas.
  - Ordenador. Utilizado para la ejecución del sistema de gestión de vídeo distribuido.
- Software:
  - VAPIX[36], API de las cámaras IP pertenecientes a la compañía Axis Communications[3]. Permiten establecer comunicación con las cámaras de red y se encuentra integrada en todos los modelos de dicha empresa.
  - Libcurl[21], biblioteca para la transferencia y recepción de archivos a través ciertos de protocolos de red. Posibilita la transferencia de imágenes de la cámara IP al equipo.
  - OpenCV[28], biblioteca dirigida al procesamiento de imágenes.
  - CoolBOT[6], framework o marco de programación orientado a componentes para sistemas robóticos. El sistema es desarrollado sobre este marco de programación.
  - ENCARA2[9], biblioteca para la detección de caras en imágenes. Esta biblioteca permite demostrar la utilidad que el material desarrollado puede aportar.

El hardware y software mencionado constituye el material principal del trabajo. La totalidad de los elementos utilizados se encuentra especificado en el capítulo "requisitos" del presente documento, sección 7.

4.3.2. Especificaciones de ejecución

El proceso llevado a cabo en la elaboración del trabajo se estructura en diferentes etapas para un desarrollo incremental. En concreto las etapas son las siguientes:

- Diseñar una serie de pre-prototipos capaces de establecer comunicación y solicitar el flujo de imágenes a una cámara IP. Para la comunicación con el dispositivo se hará uso de la API VAPIX[36] y de libcurl[21] para las transferencia de comandos VAPIX y la recepción de datos enviados por la cámara de red.

- Creación de un prototipo inicial que solvente ciertos detalles de la comunicación como son: el establecimiento de una conexión inicial con la cámara IP, autentificación y extracción de imágenes de los datos recibidos. Se implementarán diferentes funciones que resuelvan estos aspectos.
- Desarrollar una infraestructura software que integre las cámaras IP en el marco de programación CoolBOT[6]. Para ello se adaptará el material desarrollado en los pre-prototipos, petición del flujo de imágenes, y en el prototipo, establecimiento inicial de conexión con autentificación y el tratamiento de los datos para la extracción de imágenes.
- Desarrollar una infraestructura software para el almacenamiento en vídeo de las imágenes de una cámara IP, que permita el acceso offline a las secuencias grabadas por el sistema de vigilancia. Su implementación se llevará a cabo nuevamente en el framework CoolBOT[6] y utilizará la integración creada en la etapa anterior para la obtención de las imágenes.
- Desarrollar un sistema distribuido de vídeo que permita el acceso, reproducción y gestión de los vídeos grabados por la red de cámaras IP de manera integrada. Este sistema será creado bajo el marco de programación CoolBOT[6] y utilizará los vídeos generados por el software de la etapa anterior.

Cada una de las etapas de la ejecución del trabajo se encuentran explicadas con un mayor grado de detalle en el capítulo "diseño e implementación" del presente documento, sección 10.

4.4. Pliego de cláusulas administrativas

El presupuesto que se establece para el trabajo asciende a un máximo de 13590 euros. A continuación se puede ver un desglose de las cantidades que conforman esta cifra:

Recurso	Coste
Programador (250 horas)	35 euros/hora.
Cámaras IP (2 unidades)	539 euros (cada unidad).
Cableado ethernet (2 unidades)	6 euros (cada unidad).
Total	9840 euros (IGIG incl.).

Tabla 4.1: Tabla correspondiente al presupuesto asociado al trabajo.

El gasto en cámaras IP varía en función del modelo y la cantidad de cámaras escogidas. El cálculo anterior corresponde a dos cámaras de red del modelo utilizado en este TFG (Axis 211W).

4.5. Licencia

El presente trabajo se distribuye bajo licencia GNU GPL[23] (GNU General Public License) versión 3. Al ser utilizado, el usuario no adquiere ningún poder ni titularidad sobre el software que contiene. Sin embargo, podrá usarlo, distribuirlo, estudiarlo y modificarlo siempre y cuando así lo desee, tal y como establece este tipo de licencia.

Capítulo 5

Aportaciones

En los últimos años los sistemas de videovigilancia han experimentado un enorme incremento ya que permiten monitorizar el comportamiento o actividades, generalmente de personas, en un espacio físico limitado.

La principal finalidad de los sistemas de videovigilancia es la seguridad pues resultan especialmente útiles en la prevención e investigación de posibles delitos. Estos sistemas están formados por una red de cámaras desplegadas sobre el lugar que se desea controlar.

Con el paso de los años han surgido las llamadas cámaras IP o cámaras de red. Estos dispositivos han revolucionado la videovigilancia debido a que su despliegue es más sencillo y económico que las tradicionales cámaras, puesto que aprovechan el cableado de comunicación de datos ya existente al utilizar redes de comunicación IP[17]. En la actualidad la mayoría de modelos dispone de conectividad inalámbrica, lo que elimina el cableado y facilita su emplazamiento en zonas difíciles.

Con la elaboración de este trabajo se aporta, en nuestra opinión, un software de fácil manejo para la gestión de vídeo distribuido en sistemas de videovigilancia conformados por redes de cámaras IP. Así pues presenta una serie de herramientas capaces de:

- Posibilitar el acceso a una determinada cámara IP del sistema, configurada correctamente, desde cualquier lugar con acceso a internet y en cualquier momento para visualizar sus imágenes en tiempo real.
- Permitir la grabación de las imágenes de una determinada cámara IP del sistema en vídeo, de tal forma que sea posible una posterior reproducción en diferido.
- Crear un sistema distribuido que permita un acceso simultáneo a los vídeos grabados por las diferentes cámaras IP que conforman el sistema de videovigilancia. Este sistema distribuido ofrece la capacidad de reproducir múltiples

vídeos conforme al instante de tiempo en que fueron grabados, de manera que múltiples vídeos grabados al mismo tiempo serán reproducidos conjuntamente.

Al desarrollar el trabajo bajo entorno GNU/Linux se aportan además herramientas relacionadas con cámaras IP a las distintas distribuciones de Linux que, por lo general, son sólo llevadas a cabo para otras plataformas.

Adicionalmente el presente TFG, gracias a su diseño y desarrollo, es ampliable. Por ello puede constituir la base de futuros proyectos técnicos y de investigación relacionados con el tratamiento de imágenes procedentes de cámaras de red. Algunas funcionalidades posibles son la detección de personas, identificación de matrículas, entre otros.

Todo el material desarrollado quedará a disposición del instituto universitario SIANI para posibles trabajos de investigación relacionados con cámaras IP. Asimismo, se encontrará disponible para todo el mundo en la página web del proyecto CoolBOT[6].

Capítulo 6

Normativa y Legislación

A continuación se incluye la legislación vigente que afecta al presente Trabajo Fin de Grado.

6.1. Normativa

6.1.1. Ley orgánica de protección de datos

El primer artículo de la **Ley Orgánica de Protección de Datos de Carácter Personal** (LOPD)[38] dispone: "La presente Ley Orgánica tiene por objeto garantizar y proteger, en lo que concierne al tratamiento de los datos personales, las libertades públicas y los derechos fundamentales de las personas físicas, y especialmente de su honor e intimidad personal y familiar".

El su artículo 2.1 se especifica que la presente Ley Orgánica será de aplicación a los datos de carácter personal registrados en soporte físico, que los haga susceptibles de tratamiento, y a toda modalidad de uso posterior de estos datos por los sectores público y privado.

En el artículo 3 define en su letra a) los datos de carácter personal como "cualquier información concerniente a personas físicas identificadas o identificables", como puede ser en caso del presente TFG las imágenes o vídeos de cámaras IP.

Nuevamente en el artículo 3 define en su letra c) el tratamiento de datos como aquellas "operaciones y procedimientos técnicos de carácter automatizado o no, que permitan la recogida, grabación, conservación, elaboración, modificación, bloqueo y cancelación, así como las cesiones de datos que resulten de comunicaciones, consultas, interconexiones y transferencias".

En definitiva, el principal objetivo de la LOPD[38] es regular el tratamiento de los datos y ficheros, de carácter personal, independientemente del soporte en el cual sean tratados, los derechos de los ciudadanos sobre ellos y las obligaciones de

aquellos que los crean o tratan.

Las cámaras IP, y la videovigilancia en general, permite la captación, y en su caso la grabación, de información personal en forma de imágenes. Cuando su uso afecta a personas identificadas o identificables esta información constituye un dato de carácter personal a efectos de la presente Ley Orgánica 15/1999 (LOPD).

6.1.2. Instrucción 1/2006, de 8 de noviembre

Con el fin de adecuar los tratamientos de imágenes con fines de vigilancia a los principios de la Ley Orgánica de Protección de Datos[38] y garantizar los derechos de las personas cuyas imágenes son tratadas por medio de tales procedimientos se dictó la **Instrucción 1/2006, de 8 de noviembre[39]**, de la Agencia Española de Protección de Datos.

El primer artículo define el ámbito de aplicación: "la presente Instrucción se aplica al tratamiento de datos personales de imágenes de personas físicas identificadas o identificables, con fines de vigilancia a través de sistemas de cámaras y videocámaras" .

Además, el primer artículo en su sección 3 también especifica que "no se considera objeto de regulación de esta Instrucción el tratamiento de imágenes en el ámbito personal y doméstico, entendiéndose por tal el realizado por una persona física en el marco de una actividad exclusivamente privada o familiar."

Diferentes artículos de esta instrucción regulan la videovigilancia, entre sus obligaciones están:

- El artículo 3 indica que los responsables que cuenten con sistemas de videovigilancia deberán cumplir con el deber de información previsto en el artículo 5 de la Ley Orgánica 15/1999, de 13 de diciembre 6.1.1. A tal fin deberán colocar, en las zonas videovigiladas, al menos un **distintivo informativo** tanto en espacios abiertos como cerrados. Además, se deberá tener a disposición de los interesados la información relativa al propietario de los ficheros y el lugar donde el ciudadano puede ejercitar sus derechos.
- El artículo 4.3 advierte que las cámaras instaladas en espacios privados no podrán obtener imágenes de espacios públicos salvo que resulte imprescindible para la finalidad de vigilancia pretendida, o resulte imposible evitarlo por su ubicación.
- El artículo 6 determina que las imágenes no deberán almacenarse por un plazo superior a un mes (**30 días**).
- El artículo 7.1 especifica que la creación de un **fichero de imágenes** deberá ser notificado previamente a la Agencia Española de Protección de Datos, para su inscripción en el Registro General.
- El artículo 9 determina que el responsable deberá adoptar las medidas técnicas y organizativas necesarias que garanticen la **seguridad de los datos** y eviten su alteración, pérdida, tratamiento o acceso no autorizado. Por ello las cámaras deberán estar protegidas mediante usuario y contraseña.

En definitiva, esta instrucción comprende la grabación, captación, transmisión, conservación, y almacenamiento de imágenes, incluida su reproducción o emisión, así como el tratamiento que resulte de los datos personales relacionados con ellas. No obstante, se excluyen de ella los datos personales grabados para uso doméstico y el tratamiento de imágenes por parte de las fuerzas y cuerpos de seguridad.

6.1.3. Directiva 95/46/CE

A **escala europea** la **directiva 95/46/CE**[8] constituye el texto de referencia en materia de protección de datos personales. Su aplicación concierne a los datos tratados por medios automatizados, así como a los contenidos en ficheros no automatizados. Sin embargo, no será aplicable al tratamiento de datos efectuado por una persona en el ejercicio de sus actividades particulares.

La presente Directiva tiene como objetivo proteger los derechos y las libertades de las personas en lo que respecta al tratamiento de datos personales, estableciendo principios para determinar la licitud de dicho tratamiento. Dichos principios son los siguientes:

- Los datos personales serán **tratados de manera lícita** y recogidos con **fines determinados**, explícitos y legítimos.
- El tratamiento de datos personales **sólo podrá efectuarse con el consentimiento del interesado**, salvo excepciones como: el cumplimiento de una obligación jurídica, protección del interés vital del interesado, cumplimiento de misión de interés público, etc.
- Deberá **prohibirse el tratamiento de datos** personales que revelen el origen racial o étnico, opiniones políticas, convicciones religiosas y la pertenencia a sindicatos, así como el tratamiento de los datos relativos a la salud o a la sexualidad.
- El responsable deberá facilitar **datos del tratamiento** (identidad, finalidad del tratamiento, destinatarios de los datos, etc) a la persona de quien se recaben los datos.
- Se podrá limitar el alcance de los principios anteriores con objetivo de salvaguardar, por ejemplo, la seguridad del Estado o la seguridad pública.
- El interesado **tendrá derecho a oponerse** a que los datos que le conciernen sean objeto de tratamiento.
- El responsable del tratamiento deberá aplicar las medidas adecuadas para **proteger los datos personales** contra la destrucción, accidental o ilícita, la pérdida accidental, la alteración, la difusión o accesos no autorizados.
- El responsable del tratamiento **notificará a la autoridad de control nacional** con anterioridad a la realización del tratamiento.

Al igual que la Ley Orgánica de Protección de Datos (LOPD)[38], esta Directiva de referencia europea afecta a las cámaras IP y a la videovigilancia en general cuando las imágenes captadas o almacenadas disponen de información personal.

6.2. Licencias

A continuación se detallan algunas de las licencias que afectan al software utilizado en el presente proyecto.

6.2.1. GNU GPL

La licencia Pública General de GNU[23] [25] (GNU General Public License) es una licencia creada por la Free Software Foundation en 1989. Está orientada a proteger la libre distribución, modificación y uso de software. Su propósito es proteger el software cubierto por esta licencia de intentos de apropiación indebida que pueda dar lugar a restricciones y libertades de los usuarios.

La licencia GPL, al ser un documento que cede ciertos derechos al usuario, asume la forma de un contrato, por lo que es usualmente denominada contrato de licencia o acuerdo de licencia.

El framework de programación **CoolBOT**[6] utilizado en el presente trabajo se encuentra bajo este tipo de licencia.

6.2.2. BSD

Las licencias BSD[22] son un tipo de licencias de software libre permisiva. La licencia original se utilizó para la Berkeley Software Distribution (BSD), un sistema operativo tipo Unix, de donde adoptó su nombre.

Esta licencia posee menos restricciones en comparación con otras como la GPL, estando muy cercana al dominio público. La licencia BSD, al contrario que otras como la GPL, permite el uso del código fuente en software no libre. Es muy similar en efectos a la licencia MIT, la cual veremos a continuación.

OpenCV[28] es un ejemplo de software que podemos encontrar bajo licencia BSD en el trabajo.

6.2.3. MIT

El tipo de licencia MIT[24] es una licencia de software utilizado por el Instituto Tecnológico de Massachusetts (MIT, Massachusetts Institute of Technology).

Esta licencia permite reutilizar el software así licenciado tanto para ser software libre como para ser software no libre, permitiendo no liberar los cambios realizados al programa original. También permite licenciar dichos cambios con licencia BSD, GPL, u otra cualquiera que cumpla las cláusulas de distribución.

En cuanto a efectos esta licencia es muy parecida a la licencia BSD[22] ya mencionada con anterioridad.

Un ejemplo de software que emplea esta licencia es la librería **libcurl**[21].

Capítulo 7

Requisitos

Para poder conseguir los objetivos marcados inicialmente del proyecto se ha hecho uso, durante el desarrollo de éste, de una serie de recursos. Se han desglosado en dos grupos: hardware y software.

7.1. Hardware

A nivel hardware las necesidades fueron las siguientes:

1. **Cámara IP Axis, modelo 211W**: 2 unidades.
2. **Cables ethernet**.
3. **Redes TCP/IP fijas e inalámbricas**.
4. **Ordenador**: En concreto se ha utilizado un ordenador portátil con CPU Intel Core I7 a 1.6 Ghz, memoria ram de 4Gb y una tarjeta gráfica ATI.

7.2. Software

Durante el desarrollo de este trabajo se han utilizado un gran número de herramientas software. Todas ellas compatibles con un sistema operativo GNU/Linux. Concretamente la distribución utilizada ha sido:

- Ubuntu 12.04, en su versión de 32 bits.

A continuación se enumeran las herramientas más significativas utilizadas durante el desarrollo de este trabajo clasificadas en diferentes apartados.

- Librerías y frameworks:
  - **VAPIX**[36]. Interfaz de Programación de Aplicaciones (API) perteneciente a la empresa Axis Communications[3] para el acceso y comunicación con las cámaras IP de su compañía. La versión 2.0 es la empleada, debido a su compatibilidad con el modelo de cámara IP utilizada en el trabajo.
  - **Libcurl**[21]. Biblioteca/API para la transferencia y recepción de archivos con sintaxis URL[35]. Soporta una gran cantidad de protocolos de red entre los que están HTTP[14] (Hypertext Transfer Protocol) y HTTPS[15] (Hypertext Transfer Protocol Secure).
  - **OpenCV**[28]. Biblioteca dirigida principalmente para el procesamiento de imágenes en tiempo real. Es una herramienta multiplataforma que proporciona diferentes APIs en múltiples lenguajes de programación. En este caso concreto fue utilizada su API en C++.
  - **CoolBOT**[6]. Framework o marco de programación orientado a componentes para sistemas robóticos, inspirado en el paradigma CBSE[40] (Component Based Software Engineering). Ha sido diseñado y desarrollado en el Instituto Universitario SIANI y en el Departamento de Informática y Sistemas de la ULPGC.
  - **GTK+**[13] (GIMP Tool Kit). Conjunto de bibliotecas multiplataforma para el desarrollo de interfaces gráficas de usuario (GUI). Está disponible en varios lenguajes de programación, de entre los cuales se ha utilizado C++ en el presente trabajo.
  - **ENCARA2**[9]. ENCARA2 es una biblioteca para la detección de múltiples caras en vídeo. Ha sido diseñado y desarrollado en el Instituto Universitario SIANI y en el Departamento de Informática y Sistemas de la ULPGC.

- Entornos de desarrollo software y lenguajes de programación:
  - **KDevelop**[19]. Entorno de desarrollo integrado para lenguaje C++. Incluye soporte para CMake[4].
  - **Git**[12]. Software de control de versiones utilizado para el manejo de proyectos pensando en la rapidez y la eficiencia.
  - **CMake**[4] (Cross Platform Make). Herramienta multiplataforma de generación o automatización de código. CMake es una suite separada y de más alto nivel que el sistema make común de Unix.
  - **Pkg-config**[29]. Software que provee una interfaz unificada para recopilar flags de compilación y enlazado de bibliotecas instaladas. Aunque originalmente fue diseñado para Linux actualmente es multiplataforma.
  - **g++**[11]. Compilador GNU del lenguaje C++.
  - **C++**[37]. Lenguaje de programación C++. Este ha sido el principal lenguaje utilizado a lo largo de todo el trabajo.

- Software documental:
  - **Kile**[20]. Editor de TeX/LaTeX. Se encuentra disponible para múltiples plataformas.
  - **TeX Live**[31]. Distribución de TeX/LaTeX, está por defecto en algunos sistemas Linux como Ubuntu[33].
  - **Inkscape**[16]. Programa de edición de gráficos vectoriales de código abierto.
  - **ArgoUML**[1]. Herramienta de código abierto para el modelado de diagramas UML[34].

Capítulo 8

Metodología y Plan de Trabajo

8.1. Proceso unificado

Como metodología de desarrollo se utilizaron las técnicas de desarrollo software propias del **Proceso Unificado**[41] (Unified Process). El Proceso Unificado proporciona un marco de desarrollo genérico adaptable a proyectos específicos. Posee una serie de características:

1. **Iterativo e Incremental:** El Proceso Unificado está compuesto por cuatro fases distintas: Inicio, Elaboración, Construcción y Transición. En cada una de las fases se hará un determinado número de iteraciones con el objetivo de mejorar e incluir mejoras de funcionalidades dando lugar a un incremento del proyecto en desarrollo.

2. **Dirigido por los casos de uso:** Por medio de los casos de uso se especificarán en las iteraciones los requisitos funcionales y los contenidos.

3. **Centrado en la arquitectura:** En el desarrollo del software no se usará un único modelo que describa todo el sistema. Se optará por un enfoque con múltiples modelos y vistas que describan la arquitectura de software del sistema.

4. **Enfocado en los riesgos:** Se requiere identificar los riesgos críticos en una etapa temprana. De este modo los resultados de cada iteración deben seleccionarse en un orden que asegure que los riesgos principales son considerados primero.

A continuación se puede observar de forma gráfica, en la figura 8.1, la evolución típica de un proyecto que sigue el Proceso Unificado.

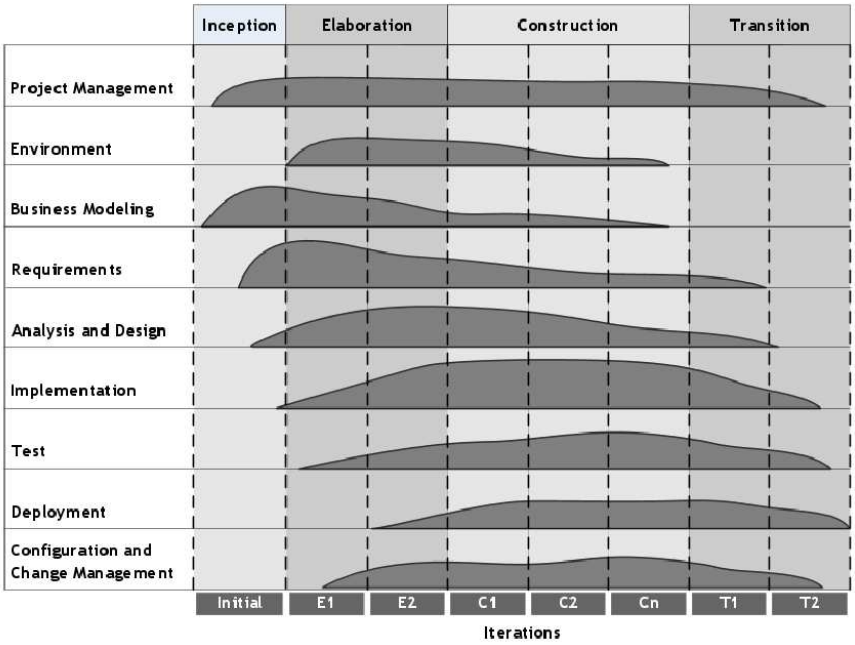

Figura 8.1: Evolución de las diferentes etapas de un proyecto desarrollado mediante el Proceso Unificado[41].

8.2. Plan de trabajo

En relación al plan de trabajo a llevar a cabo, las etapas a cubrir durante el desarrollo del TFG propuesto fueron las siguientes:

1. **Integración de cámaras IP**. Estudio y análisis de las interfaces de control y acceso a las cámaras IP a integrar en el proyecto. Diseño, desarrollo e implementación de infraestructura software (bibliotecas en C++) para su integración en el prototipo final del sistema a desarrollar.

2. **Almacenamiento distribuida de vídeo**. Estudio y análisis de bibliotecas y formatos de vídeo con licencia "libre" disponibles para soportar el almacenamiento y gestión de vídeo grabado por la cámaras integradas en el sistema. Diseño, desarrollo e implementación de infraestructura software para su integración en el prototipo final del sistema a desarrollar.

3. **Diseño y desarrollo de prototipo, y pruebas**. Desarrollo del prototipo final del sistema a desarrollar utilizando la infraestructura software desarrollada en las etapas previas. Prueba del prototipo.

4. **Documentación y defensa**. Confección del documento de trabajo de fin de grado a partir de toda la documentación realizada en las diferentes etapas en las que se ha organizado. Preparación de la defensa.

Una estimación aproximada del tiempo invertido en el desarrollo del TFG, en base a las tareas de cada etapa, es la siguiente:

Etapas	Dedicación estimada
Etapa 1: Integración de las cámaras IP	75 horas
Etapa 2: Almacenamiento distribuido de vídeo	75 horas
Etapa 3: Diseño y desarrollo de prototipo de pruebas	100 horas
Etapa 4: Documentación y defensa	50 horas
Dedicación total estimada	300 horas

Tabla 8.1: Tabla dedicación temporal estimada en cada etapa.

Capítulo 9

Análisis del Problema

En este capítulo se presentarán las tareas de análisis llevadas a cabo durante la realización de este proyecto. Concretamente se realiza un estudio previo del problema y posteriormente se describen en detalle una serie de recursos y herramientas que han sido utilizadas a lo largo del mismo. En las descripciones de recursos y herramientas se explica su funcionamiento, así como se incide en las ventajas que presentan, argumentando su elección.

9.1. Estudio del problema

El principal objetivo propuesto inicialmente para este Trabajo Fin de Grado es la **integración de cámaras IP** en el framework de programación **CoolBOT**. CoolBOT es un framework o marco de programación orientado a componentes para sistemas robóticos basado en el paradigma CBSE[40] (Component Based Software Engineering). CoolBOT ha sido diseñado y desarrollado en el Instituto Universitario SIANI y en el Departamento de Informática y Sistemas (DIS). En la actualidad es utilizado por miembros, colaboradores y estudiantes de ambas instituciones pertenecientes a la Universidad de Las Palmas de Gran Canaria.

Para lograr esta integración es necesario un estudio y análisis previo de las interfaces de control y acceso a las cámaras IP, así como de una herramienta software para transferencias en red. En la *fase inicial* de desarrollo se han creado algunos programas para aprender a establecer comunicaciones con cámaras de red. A medida que avanza el desarrollo se crea un *prototipo inicial*, fuera de CoolBOT, capaz de conectarse a la cámara IP y obtener de ella sus imágenes. Estas primeras etapas afrontarán los problemas que surjan durante su implementación, de tal forma que cuando esté finalizado se integre en CoolBOT con garantías de que todo funciona correctamente.

Una vez conseguida la integración de cámaras IP en CoolBOT, se ha marcado el objetivo de **almacenar en vídeo** sus imágenes. Con este fin se estudia y analiza

una biblioteca para el almacenamiento y gestión de vídeo. En el desarrollo de la infraestructura software se ha utilizado nuevamente CoolBOT, aprovechando la integración ya creada en el paso anterior para cámaras IP.

El último objetivo marcado es la creación de un **sistema de vídeo distribuido** que permita la gestión y reproducción simultánea de vídeos generados por cámaras IP. Este sistema respeta la temporalidad de los vídeos, de tal forma que varios vídeos que han sido grabados en el mismo instante de tiempo puedan ser vistos a la vez. En su desarrollo se ha utilizado CoolBOT, al igual que la biblioteca gráfica GTK+[13] para el diseño de una interfaz gráfica (GUI) cómoda y sencilla para el usuario.

A continuación se detalla con más profundidad cada uno de los objetivos, siguiendo los pasos tomados en el análisis para cada caso en particular.

9.1.1. Integración de cámaras IP

Como se ha comentado, las **cámaras IP** son dispositivos que permiten la visualización de imágenes en la zona que se esté monitorizando, con la peculiaridad de que los datos son transmitidos a través de la red (intranet o internet), ya sea a través de una conexión de tipo ethernet o de manera inalámbrica. Esta peculiaridad implica que, de forma previa, haya que analizar la manera de comunicarse con las cámaras a través de la red.

Al utilizar cámaras pertenecientes a la compañía Axis Communications[3], se ha aprovechado la API que proporciona dicho fabricante. La API, denominada VAPIX[36], permite una interacción con la cámara de red desde cualquier equipo que tenga acceso a ella (ya sea desde intranet o internet) mediante el protocolo HTTP[14] (Hypertext Transfer Protocol). Con el fin de conseguir una correcta comunicación con la cámara IP es imprescindible estudiar con cierto detalle esta API, estudio reflejado en la sección 9.3.1.

Un dato importante que se obtiene en el transcurso del estudio de VAPIX[36], reflejado en posteriores secciones, es que en la retransmisión de vídeo la cámara IP envía imagen a imagen por la red, separándolas entre sí con una cabecera. No obstante, cuando envía una imagen ésta no está completa, es decir, **los datos de la imagen son enviados por trozos**.

Una vez terminado el estudio de la interfaz de programación VAPIX[36] ya se conocen los comandos necesarios para comunicarse con cámaras IP. Sin embargo, tanto el envío de los comandos como la recepción de los datos que provee la cámara son realizados a través del protocolo de red HTTP[14]. Esto provoca que para enviar y recibir datos sea necesario hacer uso de alguna herramienta que permita una comunicación por dicho protocolo.

Tras un análisis de las diferentes bibliotecas que permiten transmitir datos vía HTTP se decidió hacer uso de libcurl[21], una biblioteca que soporta gran cantidad de protocolos que ofrece sencillez de uso y una abstracción de la capa de red. Como ocurriera con VAPIX[36] se hace necesario un estudio previo de esta biblioteca para

utilizar las herramientas adecuadas, estudio reflejado en la sección 9.3.2.

Una vez estudiadas estas dos herramientas; VAPIX[36] y libcurl[21], es posible crear programas pre-prototipos como primera aproximación para establecer un comunicación con la cámara IP y aprender el manejo de ambas herramientas.

Tras la implementación de los pre-prototipos se plantea la creación de un prototipo inicial que se conecte a la cámara IP e interactúe con ella.

9.1.1.1. Prototipo inicial

Este prototipo representa la base de todo el trabajo. En él se realiza una primera integración con cámaras IP para posteriormente facilitar la creación de la infraestructura software. El objetivo final de este prototipo es crear una librería capaz de conectarse a una cámara IP y obtener de ella las imágenes que reproduce, guardándolas en archivos.

Para ello se sigue una serie de etapas que permiten el desarrollo incremental, añadiendo en cada etapa funcionalidades nuevas o ampliando las ya creadas. Así se sigue la metodología del **proceso unificado**[41]. Las etapas diseñadas son las siguientes:

- **Etapa 1**. En la primera etapa se genera una conexión inicial con la cámara IP que permita identificar posibles errores de la conexión como autentificación errónea, dispositivo no encontrado, etc. Por este motivo se procederá a la creación de una función que establezca dicha conexión y detecte posibles errores.
- **Etapa 2**. Tras comprobar que la conexión es correcta se solicita a la cámara IP un "stream" o flujo de imágenes. Para ello se utilizará el comando VAPIX[36] correspondiente y el software libcurl[21] para su transmisión. Se tomará como referencia para esta etapa el pre-prototipo 1 creado en la fase inicial, sección 10.1.1.
- **Etapa 3**. Creación de una función capaz de detectar una ristra de caracteres específica dentro de un bloque de datos. Esta función será necesaria para el posterior tratamiento de los datos enviados por la cámara IP.
- **Etapa 4**. Creación de un nuevo tipo de buffer capaz de almacenar los datos de una imagen y realizar un manejo eficiente de la memoria dinámica. El buffer será de gran utilidad en el posterior tratamiento de los datos enviados por la cámara de red.
- **Etapa 5**. Una vez solicitado el flujo de imágenes se requiere tratar la información recibida del dispositivo de red. Con dicha finalidad se procede a la creación de una función que reciba dichos datos y que extraiga cada imagen contenida en ellos, para lo cual utilizarán la función y el buffer de las etapas 3 y 4, respectivamente. Esta nueva función ha de especificarse entre las opciones de la transferencia, etapa 2, para poder recibir los datos de la cámara IP.

Una vez terminadas todas las etapas se dispone de una librería capaz de conectarse a una cámara IP y obtener de ella las imágenes que emite en tiempo real.

9.1.1.2. Infraestructura software

Terminado el prototipo se dispone de una integración inicial para cámaras de red. El siguiente paso es la creación de la infraestructura software que **integre las cámaras IP** en el framework de programación CoolBOT[6]. Para ello se adaptará prototipo implementado en el paso anterior.

En este caso se ha escogido el framework de programación CoolBOT[6], el cual permite la creación de un sistema modular, de fácil ampliación , robusto, sencillez de diseño, entre otras características. Como cualquier herramienta necesita de un estudio previo para su correcta utilización, el estudio y análisis de CoolBOT se ve reflejado en la sección 9.3.4.

El principal objetivo en esta etapa del TFG es adaptar el prototipo inicialmente creado a CoolBOT, con el fin de crear un sistema que permita la visualización de imágenes de la cámara IP y facilite en futuras etapas la grabación en vídeo. Todas las funciones creadas en la librería del prototipo inicial seguirán utilizándose, aunque algunas de ellas serán modificadas para adaptarse a CoolBOT.

9.1.2. Almacenamiento en vídeo

Con el fin de almacenar las imágenes obtenidas de la cámara IP se inicia la búsqueda de una biblioteca de **grabación de vídeo**. En un principio se busca una biblioteca capaz de grabar en tiempo real, de tal forma que mientras esté grabando genere un archivo de vídeo reproducible al mismo tiempo que continúa la grabación. Este tipo de bibliotecas resulta interesante, sin embargo, su utilización y gestión es compleja, por ello se requiere un largo tiempo de estudio y familiarización con las mismas. Este motivo propició que se decidiera utilizar una biblioteca de grabación más sencilla que genere un archivo normal de vídeo cuando el usuario finalice la grabación.

La biblioteca finalmente escogida fue OpenCV[28], que soporta una gran cantidad de funciones para imágenes como: conversión de formatos, reducción o ampliación de una imagen, grabación de vídeo, etc. En concreto las funciones utilizadas en este trabajo se encuentran en la librería HighGui de OpenCV. Para hacer un uso correcto de esta biblioteca es necesario un estudio de OpenCV[28], aunque en este caso no es necesario un estudio tan profundo como con anteriores herramientas, análisis de OpenCV reflejado en la sección 9.3.3.

A la hora de crear la infraestructura software para el almacenamiento de vídeo se plantea crear un sistema utilizando el framework CoolBOT[6]. Esta infraestructura aprovecha el flujo de imágenes enviado por la integración de cámaras IP, creada en la etapa anterior, para su almacenamiento.

9.1.3. Sistema de vídeo distribuido

El último objetivo es la realización de un **sistema de vídeo distribuido**. El sistema a generar permitirá la reproducción de los múltiples vídeos generados, respetando el momento en el que se inició su grabación. De este modo, al respetar el inicio de la grabación, es fácil visualizar varios vídeos que coinciden en el tiempo

o la diferencia de tiempo entre uno y otro.

Un boceto del prototipo final es el que se puede ver en la figura 9.1. Cada vídeo posee su línea de tiempo y en caso de que múltiples vídeos coincidan en el tiempo se reproducen de manera simultánea.

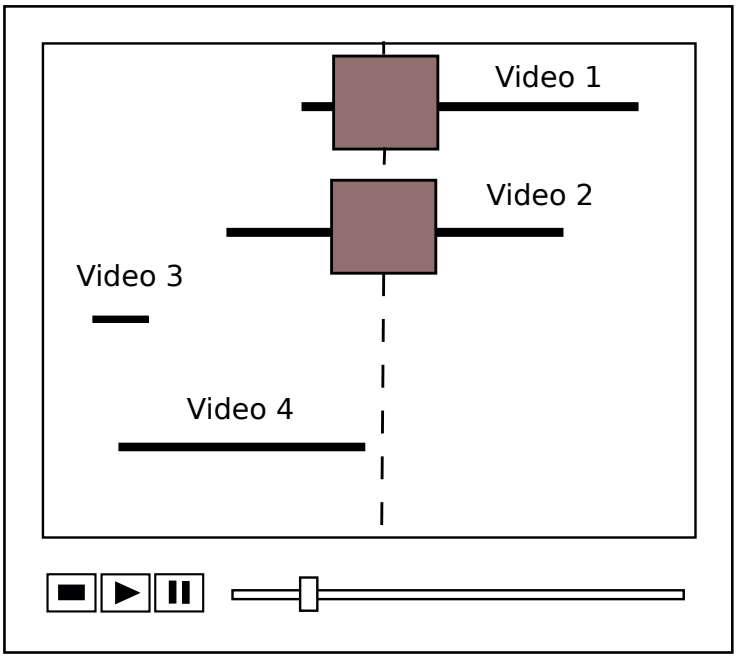

Figura 9.1: Diseño inicial.

Existen una serie de problemas que son necesarios solventar para la creación de este sistema de vídeo distribuido. Todos ellos se detallan a continuación.

1. Necesidad de **registrar el inicio de la grabación**. Debido a que se quiere situar temporalmente todos los vídeos es necesario conocer el momento exacto en el que se inició la grabación. Esta información sólo es conocida al empezar a grabar, por lo que es necesario registrarla en ese instante de tiempo. La solución propuesta es almacenar el tiempo inicial en el propio nombre del vídeo con un formato especial.

2. **Gestionar los rangos temporales**. Es necesario realizar una gestión inteligente de los rangos de tiempo de cada vídeo para poder determinar en un instante de tiempo concreto qué vídeos poseen imágenes.

3. Creación de **una interfaz gráfica** que muestre en pantalla la información de manera visual, así como las imágenes cuando corresponda.

4. **Uso de controles de reproducción integrada** (play, stop, ...). Detectar el uso de dichos controles y actuar en consecuencia. Cada control puede requerir una acción diferente, desde reiniciar la línea de tiempo hasta ir a un instante de tiempo concreto.

Como las anteriores etapas del proyecto, este sistema de vídeo distribuido será creado con el framework de programación CoolBOT[6]. Adicionalmente en esta etapa se crea una interfaz gráfica sencilla y cómoda para el usuario. GTK+[13] es el software con el que se desarrolla esta interfaz gráfica de usuario, al ser la librería que utiliza CoolBOT para la creación de interfaces gráficas en sus *vistas*.

9.2. Recursos hardware

9.2.1. Cámaras IP

Una cámara IP es una cámara que emite las imágenes directamente a la red (ya sea intranet o internet) sin necesidad de un ordenador. Internamente este tipo de dispositivos tienen incorporado un microordenador especializado en ejecutar aplicaciones de red. Además, posee su propia dirección IP y se conecta a la red como cualquier otro dispositivo de red.

Su principal función es la compresión y el envío de vídeo a través de la red, a dispositivos que así lo soliciten. En algunos casos disponen de funciones adicionales como: activación a través de sensores, control remoto para mover la cámara, etc.

Hoy en día la principal utilidad de las **cámaras IP** o **cámaras de red** es la videovigilancia, en gran medida gracias a que permiten la visión de vídeo en directo desde cualquier lugar con conexión a internet. Evidentemente el acceso a estas imágenes está totalmente restringido de tal forma que sólo las personas autorizadas puedan verlas.

En el mercado existen multitud de compañías que se encargan de la fabricación de cámaras IP para vigilancia. Sin embargo, Axis Communications[3] es probablemente la empresa líder en este mercado, no en vano fue la creadora de la primera cámara IP en 1996.

Además de tener productos ya muy extendidos, Axis también proporciona una API con la que poder interactuar con sus cámaras. Estas razones nos llevan a utilizar una cámara IP Axis en el trabajo, pudiendo comunicarnos con ella de manera sencilla y sabiendo además que podrá ser ejecutado por un amplio número de personas en todo el mundo.

9.2.1.1. Cámara IP Axis 211W

El modelo concreto utilizado durante el desarrollo del presente TFG ha sido la cámara IP Axis 211W (figura 9.2) la cual dispone de las siguientes características generales:

- Flexible: soporta conexión por ethernet y también conexión inalámbrica.
- Seguridad de red: ofrece niveles de acceso multiusuario, protección por contraseña, filtrado de direcciones IP y cifrado HTTPS.
- Soporta el protocolo de internet versión 6 (IPv6) además de la versión 4.
- Soporta control mediante eventos.

- Compatibilidad con la interfaz de programación de aplicaciones (API) de Axis[3], VAPIX[36].

Figura 9.2: Cámara IP Axis modelo 211W

9.3. Herramientas software

9.3.1. VAPIX

Las cámaras IP de la compañía Axis[3] proporcionan una interfaz de programación de aplicaciones (API) denominada VAPIX[36]. VAPIX está, en gran medida, basada en el protocolo HTTP y permite la petición de imágenes, control de funciones propias de la cámara (zoom, enfoque,..), ver y modificar parámetros internos del dispositivo (resolución, frames por segundo,..) y mucho más.

Actualmente VAPIX[36] posee dos versiones: la versión 3, actualizada para nuevos dispositivos, y la versión 2. Dependiendo del modelo se podrá hacer uso de una u otra. En concreto, el modelo de cámara 211W utilizado en este trabajo sólo soporta la versión 2, es por ello que esa será la utilizada. Aclarar que las funciones utilizadas de VAPIX en este proyecto están soportadas en ambas versiones.

La versión 2 de VAPIX se compone de tres elementos principales:

- **HTTP API**. API que contiene los comandos o controles, bajo el protocolo HTTP, para la mayoría de opciones como petición de imágenes, creación/modificación/eliminación de parámetros, control de zoom/enfoque/movimiento, entre muchas otras funciones.
- **Parameter specification** (especificación de parámetros). Establece los parámetros de configuración que contiene internamente la cámara IP. Por ejemplo: lista de usuarios, configuración actual de red, etc. Estos parámetros son modificables con controles de la API HTTP, siempre y cuando se dispongan permisos para ello.

- **RTSP API**. API que posee los controles, bajo el protocolo RTSP[30] (Real Time Streaming Protocol), para el control de vídeo con formato MPEG-4[27].

Cada una de estas tres partes que componen VAPIX[36] están documentadas en la página oficial del fabricante Axis[3].

En el estudio de esta interfaz de programación se centrará en el primero de los componentes (HTTP API), aunque en primer lugar será necesario conocer los parámetros (parameter specification).

9.3.1.1. Parámetros

Las principales características de las cámaras IP Axis son configuradas por parámetros. Estos parámetros por lo general se encuentran en diferentes archivos de configuración dentro de la memoria flash del dispositivo. Existe una gran variedad de parámetros, sin embargo, algunos de ellos sólo están disponibles en ciertos modelos de cámaras IP.

Los grupos de parámetros más importantes son:

- *General*. En este grupo se especifican los parámetros de configuración básicos como parámetros de red (configuración de red actual, configuración por defecto, etc), parámetros para controlar los archivos de registro (archivos de log) , parámetros para el control de protocolos (si son habilitados y sus configuraciones), hora del sistema, entre muchos otros.
- *Imagen*. Aquí se especifican los parámetros relacionados directamente con la imagen. Por poner un ejemplo, algunos parámetros concretos pueden ser la resolución de la imagen o su compresión.
- *PTZ*: pan, tilt, zoom (enfoque, inclinación y zoom). Este grupo contiene los parámetros propios para el enfoque, inclinación y zoom de la cámara IP, pero sólo están disponibles en modelos de cámaras IP móviles.
- *Wireless* (inalámbrico). En él se encuentran los parámetros para la configuración de red inalámbrica.

Esta es una selección de los grupos más importantes, pero existen muchos otros. Es necesario aclarar que cada parámetro posee un nivel de seguridad para garantizar que los datos son vistos o modificados por usuarios con permisos para ello. Por ejemplo, si se desea modificar la dirección IP que toma la cámara de red por defecto es necesario ser un administrador.

9.3.1.2. Comandos

Los **controles o comandos** que permiten una comunicación con la cámara IP se encuentran en el componente HTTP API. Como se ha comentado, estos comandos son ejecutados bajo el protocolo HTTP (Hypertext Transfer Protocol) y siguen el método estándar de comunicación entre cliente y servidor CGI[5], Common Gateway Interface.

De manera interna la cámara IP dispone de archivos CGI, estructurados en directorios, que permiten la interacción con los diferentes grupos de parámetros ya vistos en la sección anterior. Por ejemplo, el archivo "video.cgi" permite la modificación de parámetros relacionados con el vídeo (resolución, compresión, etc).

De forma general los comandos de VAPIX[36] siguen una **sintaxis** CGI URL como la siguiente:

```
http://<servername>/axis-cgi/<subdir>[/<subdir>...]/<cgi>.<ext>[?<parameter
    >=<value>[&<parameter>=<value>...]]
```

Los elementos que se encuentran entre corchetes ('[' y ']') son opcionales, mientras que aquellos que se encuentran entre los símbolos '<' y '>' serán reemplazados por los elementos que se detallan entre los mismos. En este caso los elementos son:

- *servername*: dirección IP o dominio de la cámara de red.
- *subdir*: subdirectorio donde se encuentre un archivo CGI para realizar diferentes acciones.
- *cgi*: nombre del archivo CGI.
- *ext*: extensión del archivo CGI (".cgi" por lo general).
- *parameter*: parámetro que se desea utilizar.
- *value*: valor con el que modificar un parámetro o acción a realizar con él.

A continuación se muestra un comando de ejemplo.

```
http://<servername>/axis-cgi/admin/param.cgi?action=list&group=Network
```

Este comando de ejemplo pide a la cámara IP listar todos los parámetros existentes, junto con sus valores actuales, que pertenezcan al grupo "Network" (configuración de red).

En la **respuesta a los comandos** enviados, el servidor web que incorpora la cámara de red utiliza los códigos de estado HTTP al inicio de los datos para determinar si la petición fue exitosa o si bien ocurrió algún fallo específico. Algunos de los códigos de estado HTTP son los que podemos encontrar en la tabla 9.1.

El formato con el que envía esta información es:

```
HTTP/1.0 <HTTP code> <HTTP text>\r\n
```

"HTTP code" corresponde al código de estado y "HTTP text" al texto indicado en la tabla 9.1.

Código HTTP	Texto	Descripción
200	OK	La petición ha sido exitosa.
204	No content	Se ha recogido la petición, pero no hay información nueva que enviar.
400	Bad request	La petición se ha realizado de manera errónea o ha sido imposible de realizar.
401	Unauthorized	Se requiere autentificación o la autorización ha sido denegada.
404	Not found	El servidor no ha podido encontrar nada que coincida con la petición.

Tabla 9.1: Codigos de estado HTTP.

Al igual que ocurriera con los parámetros, los comandos de VAPIX[36] se engloban en diferentes grupos.

- *General*. Contiene comandos para listar/añadir/modificar parámetros internos, añadir/modificar/eliminar usuarios con acceso a la cámara, restaurar los valores de fábrica, acceder al contenido de los archivos de registro (archivos de log), actualizar/ver la hora del sistema, entre muchos otros.
- *Imagen y vídeo*. Este grupo abarca los comandos para petición de una imagen o de vídeo, así como la modificación de parámetros propios a la imagen (resolución, compresión de la imagen, etc).
- *PTZ: pan, tilt, zoom (enfoque, inclinación y zoom)*. Aquí se encuentran las órdenes para enfocar, mover la cámara IP hacia una zona determinada o hacer zoom. Sólo disponible en algunos modelos concretos.

Estos son sólo los tres grupos más utilizados generalmente, pero existen muchos otros para audio, conexiones de entrada/salida, etc. Para familiarizarse con esta interfaz de programación se han estudiado los grupos más importantes, pero se ha hecho especial hincapié en el de "imagen y vídeo".

9.3.1.3. Comando vídeo

Conocidos ya los grupos de comandos, se accede al grupo "imagen y vídeo" para escoger una orden concreta a través de la cual la cámara IP inicie el vídeo, así como para conocer la forma en la que las imágenes son enviadas.

Entre los formatos de vídeo disponibles en esta versión de la API están MJPG[26](Motion JPEG) y MPEG-4[27]. Se ha decidido utilizar el primero de ellos (MJPG) entre otros motivos por la sencillez del formato en el que la cámara envía los datos y porque se controla en todo momento a través de la API HTTP. El formato de vídeo MPEG-4[27] es tratado bajo el protocolo RTSP[30] (Real time stream protocol).

Antes de continuar conviene definir MJPG[26]. MJPG (Motion JPEG) es un formato de vídeo en el cual cada imagen que pertenece al vídeo es comprimida de forma individual como una imagen en formato JPEG[18] (método de compresión para imágenes). En la actualidad es frecuente su utilización en cámaras IP, consolas y otros dispositivos.

El comando utilizado hace que la cámara IP envíe las imágenes que captura en tiempo real. El formato que genera VAPIX al enviar las imágenes es el de incluir al **inicio de cada imagen** JPEG una **cabecera** específica, con el fin de delimitar el inicio y fin de los datos de una imagen. A continuación se ve el formato:

```
HTTP/1.0 200 OK
Content-Type: multipart/x-mixed-replace;boundary=myboundary

--myboundary
Content-Type: image/jpeg
Content-Length: 15656

<JPEG image data>

--myboundary
Content-Type: image/jpeg
Content-Length: 14978

<JPEG image data>

--myboundary
Content-Type: image/jpeg
Content-Length: 15136

<JPEG image data>

--myboundary
.
.
.
```

En este pequeño extracto de ejemplo vemos como la cabecera incluida antes de los datos de cada imagen constan de un separador ("`--myboundary`"), el tipo de datos ("`Content-Type: image/jpeg`") y la longitud de los mismos ("`Content-Length:`"). El separador es un elemento que siempre aparece en la cabecera, no obstante, el tipo de datos y su longitud son adjuntados o no en función de la cámara IP.

Otro detalle muy importante a tener en cuenta es que el envío de todos **los datos**, tanto los de cabecera como los de la imagen, **son enviados por partes** a través de la red, es decir, que los datos de una imagen se encuentran troceados. En este caso ha de ser el receptor quien, con ayuda de las cabeceras, diferencie entre una imagen y otra al recibir los datos.

9.3.2. Libcurl

Tras el análisis de VAPIX[36] se hace imperiosa la necesidad de trabajar con los protocolos HTTP[14] (Hypertext Transfer Protocol) y HTTPS[15] (Hypertext Transfer Protocol Secure) para poder establecer comunicación con la cámara IP. Por esa razón se busca una biblioteca que permita el envío y recepción de datos a través de los comentados protocolos de red.

La biblioteca escogida para esta tarea es **libcurl**[21]. Libcurl forma parte del denominado proyecto cURL y permite la transferencia y recepción de archivos con sintaxis URL. Soporta una gran cantidad de protocolos de red como es el caso de: FTP, FTPS, HTTP, HTTPS, TFTP, SCP, SFTP, Telnet, DICT, entre muchos otros. Está disponible en una gran multitud de lenguajes de programación, aunque la biblioteca original está desarrollada en C.

Libcurl ofrece unas funciones sencillas de utilizar que abstraen al programador de los detalles de la capa de red y, además, facilita el tratamiento de los datos recibidos vía HTTP en tiempo real. Estas razones hacen que libcurl[21] sea la herramienta seleccionada para la comunicación con la cámara IP.

Durante el estudio de esta herramienta se ha observado que dispone de dos interfaces, llamadas "**easy interface**" (interfaz fácil/sencilla) y "**multi interface**" (interfaz múltiple). Ambas proporcionan un conjunto de funciones para transferencias de red. Son muy similares entre sí, aunque cada una posee sus peculiaridades. A continuación se explicará en qué consiste cada una, así como su funcionamiento.

9.3.2.1. Easy interface

La interfaz "easy" es la más básica de las que componen libcurl[21]. Se trata de una interfaz que proporciona una serie de funciones para transferencias bajo los protocolos soportados de una forma síncrona, eficiente y de fácil uso.

El conjunto de funciones que componen esta interfaz permiten el envío y recepción de datos de un recurso en red, como puede ser en este caso una cámara IP, desde cualquiera de los protocolos que soporta. Para ello necesita obligatoriamente la URL[35] (Uniform Resource Locator) para localizar el recurso.

Cuando se utiliza esta interfaz se ha de iniciar sesión con el comando correspondiente para obtener un manejador que determina una nueva sesión y las características de la misma. Este manejador es necesario para la utilización del resto de funciones.

Posteriormente se continúa estableciendo las opciones de la transferencia que se desea realizar (con el comando correspondiente). La opción obligatoria es la URL del recurso al cual conectarse, para enviar o recibir información. Existen multitud de posibles opciones a configurar entre ellas: parámetros de autentificación, método de autentificación, etc.

Una vez fijadas las opciones se inicia la transferencia. A medida que va recibiendo datos, libcurl proporciona los datos recibidos a una función. Por defecto esta función es interna a libcurl e imprime por pantalla la información recibida. No obstante, en las opciones de transferencia es posible establecer una función propia que reciba dichos datos, siempre que cumpla una serie de características.

La transferencia es continua y finaliza cuando el emisor deja de enviar datos. Una vez terminado es posible reutilizar el manejador cambiando sus atributos para una nueva transferencia o finalizar la sesión.

9.3.2.2. Multi interface

La interfaz múltiple ofrece la mismas funcionalidades que la "easy interface" y añade otras nuevas como permitir varias transferencias simultáneas por un mismo hilo de ejecución (de ahí el nombre de interfaz), es asíncrona, permite terminar una transferencia aunque el emisor continúe enviando datos, etc.

Al principio se inicia sesión con el comando correspondiente para obtener un manejador múltiple. Cada una de las transferencias que se van a realizar se tratan como en la interfaz "easy", creando manejadores individuales y fijando sus opciones. De esta forma se puede ver la interfaz múltiple como una composición de muchas interfaces simples, "easy interfaces".

El siguiente paso a dar es unificar todas las transferencias individuales que se desean realizar bajo el manejador múltiple. Una vez unificadas con la función correspondiente ya es posible iniciar las transferencias. Al contrario que ocurría con la interfaz "easy", ahora cada vez que se reciben datos es posible decidir qué acción realizar con ellos, pudiendo desde tratar la información con las funciones propias a cada transferencia individual hasta finalizar en cualquier momento la ejecución.

Cuando terminen las transferencias o se finalice voluntariamente la recepción o envío de datos es necesario terminar con todas las sesiones libcurl iniciadas, tanto múltiples como individuales.

9.3.3. OpenCV

Ante el objetivo de desarrollar una infraestructura software para la grabación, reproducción y acceso offline a las grabaciones realizadas, iniciamos una búsqueda de bibliotecas de grabación de vídeo más utilizadas.

La biblioteca finalmente escogida es **OpenCV**[28]. Ésta es una biblioteca muy expandida de visión artificial, la cual posee más de 500 funciones que abarcan una gran gama de áreas en visión por computador. Precisar que dentro de OpenCV las funciones utilizadas para este TFG son las encontradas en la librería HighGui. Esta librería suministra funciones capaces de convertir formatos de imágenes, reducir/ampliar tamaño de imágenes, posibilita la generación de vídeo, etc.

OpenCV ha sido diseñada para la eficiencia computacional y con un gran enfoque en aplicaciones de tiempo real. Originalmente fue escrito en lenguaje C optimizado, pero proporciona múltiples APIs en otros lenguajes de programación como C++, python, etc. En concreto en este TFG se utiliza su API en C++.

9.3.4. CoolBOT

El framework de programación CoolBOT[6] es la herramienta fundamental sobre la que se ha desarrollado este proyecto. La programación en CoolBOT aporta numerosas características como modularidad, facilidad de ampliación, sistema robusto, sencillez en el diseño, etc.

Debido a que el desarrollo de este TFG se realiza casi en su totalidad en CoolBOT, es imprescindible analizar en profundidad esta herramienta. Nos centraremos esencialmente en los aspectos de dicha plataforma necesarios para el presente trabajo.

9.3.4.1. ¿Qué es?

CoolBOT es un framework o marco de programación orientado a componentes para sistemas robóticos. Ha sido diseñado, desarrollado y es utilizado actualmente en el Laboratorio de Robótica y Oceanografía Computacional del Instituto Universitario SIANI y en el Departamento de Informática y Sistemas de la ULPGC. CoolBOT está inspirado en el paradigma de ingeniería del software CBSE[40], del inglés Component Based Software Engineering.

El desarrollo de esta plataforma surge de la falta de paradigmas claros y estandarizados de programación para este tipo de sistemas. Esta situación va en contra de la reutilización de código, factor clave, cuanto más en este tipo de sistemas, donde existen muchos problemas recurrentes tales como la abstracción del hardware o la computación distribuida.

CoolBOT presenta un modelo de programación basado en el concepto de **componente software**, donde los elementos que componen el software de un sistema robótico se denominan componentes. En CoolBOT se pueden encontrar **tres tipos de componentes**: *componentes*, *vistas* y *sondas* (components, views y probes). La principal diferencia entre ellos es que las *vistas* y *sondas* son componentes software "livianos" con respecto a los *componentes* CoolBOT.

Un *componente* es una unidad que implementa y aporta una determinada funcionalidad al sistema en conjunto, para lo cual coopera con otros componentes consumiendo y/o produciendo datos. Cada uno de estos *componentes* CoolBOT se ejecuta individualmente como una máquina de flujo de datos, que se encuentra inactiva esperando datos en sus entradas, activándose sólo cuando llegan dichos datos para ser procesados y, seguidamente, emitir datos procesados a través de sus salidas.

Los *componentes* de CoolBOT son modelados como un autómata de puertos (figura 9.4) y representan unidades independientes de ejecución. Estos componentes, una vez diseñados y construidos, pueden ser instanciados cuantas veces sea necesario.

Las *vistas* son componentes software que implementan el control gráfico y las interfaces de monitorización para sistemas CoolBOT, de los cuales está completamente desacoplado. Por otro lado, las *sondas* (probes) permiten implementar interfaces desacopladas para realizar operaciones de sistemas CoolBOT con aplicaciones que no están hechas en CoolBOT.

Los componentes se comunican entre ellos a través de su interfaz externa, la cual está formada por puertos de entrada y salida. Cuando se conectan, forman una conexión de puertos mediante la cual intercambian unidades discretas de información denominadas "**port packets**". Las vistas y sondas poseen una interfaz externa similar que permite su conexión con componentes. La funcionalidad del sistema

surge de la interacción entre todos los componentes del sistema, incluidas vistas y sondas.

9.3.4.2. El modelo de componente

En CoolBOT los componentes son objetos activos, es decir, unidades de computación que encapsulan los datos y realizan operaciones con ellos para obtener un determinado resultado. Se puede ver como una unidad que procesa datos tan pronto los obtiene por sus puertos de entrada (u otros medios) y envían información procesada con esos datos a través de unidades discretas de información denominadas "port packets" por sus puertos de salida.

Cada componente es modelado internamente como un autómata, figura 9.4, con puertos de entrada y puertos de salida. Este modelo permite diferenciar claramente la funcionalidad interna del componente de la interfaz externa como puede verse en la siguiente figura 9.3.

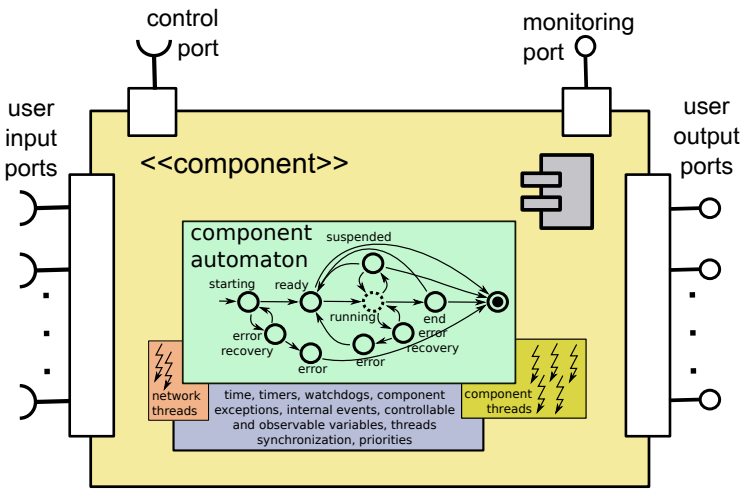

Figura 9.3: Estructura de un componente CoolBOT

Como se puede ver en la figura 9.3, internamente un componente CoolBOT aporta una serie de herramientas (timers, excepciones, etc). Destacar entre ellas el soporte de red, el cual permite comunicaciones con otros componentes que puedan encontrarse en redes diferentes de una forma transparente y desacoplada.

Externamente la estructura de un componente consta de dos puertos externos especiales: el puerto de control (superior izquierda) y el puerto de monitorización (superior derecha), los cuales permiten la interacción con un supervisor externo para el control de la ejecución de cualquier componente mediante el uso de un protocolo estándar. El resto de puertos existentes, de salida y entrada (izquierda y derecha de la figura), son definidos por quien diseñe el componente para la recolección o envío de datos a otros componentes.

9.3.4.3. Autómata por defecto

Todos los componentes que se crean en CoolBOT son modelados a partir de un autómata por defecto. Podemos ver una representación de este autómata en la figura 9.4, donde los diferentes estados se representan por un círculo y las transiciones son representadas por arcos etiquetados para indicar el evento que dispara la transición.

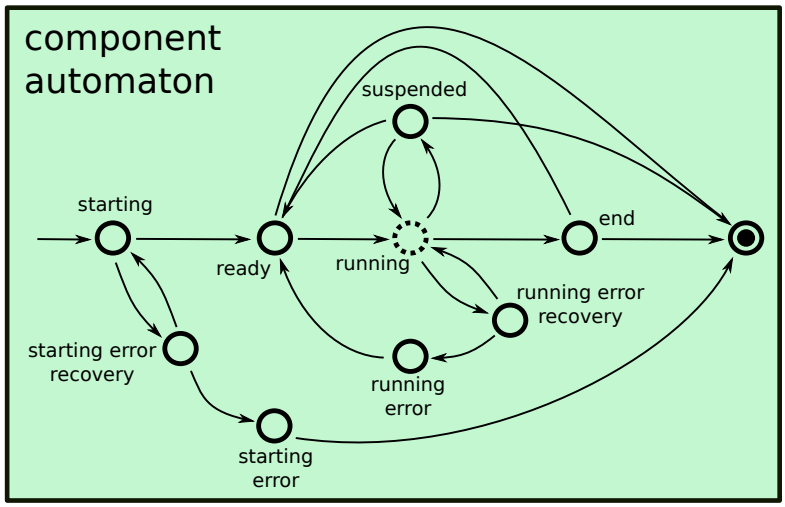

Figura 9.4: Autómata por defecto en componentes CoolBOT

Las transiciones dibujadas en la figura 9.4 corresponden a los eventos por defecto que tiene el autómata. Fundamentalmente están asociadas al tratamiento de posibles excepciones que puedan ocurrir en las diferentes etapas de ejecución, aunque también existen transiciones asociadas a la suspensión/reanudación de la ejecución.

En cuanto a los estados, podemos visualizar diferentes estados que van asociados a diferentes etapas de la vida de un componente. Los estados más destacables son:

- **Starting**: se adquieren los recursos necesarios para la ejecución del componente.
- **Ready**: estado previo a la ejecución en el que se mantiene a la espera de la orden para entrar a ejecutarse.
- **Running**: realmente no es un estado al uso, sino un metaestado. En este momento se ejecutará el **autómata diseñado por el usuario**, el cual determina la funcionalidad concreta del componente.
- **Suspended**: el componente se encuentra suspendido a la espera de que se le ordene transitar a otro estado.
- **End**: se ha finalizado correctamente la ejecución del componente y se publica el resultado, en caso de que hubiera, por el puerto de monitorización.

- **Dead**: corresponde al estado final. En él se liberan los recursos utilizados y se finaliza por completo el componente.

Además de los nombrados, existen otros estados a los cuales se accede en caso de surgir algún tipo de error en cualquiera de las etapas nombradas con anterioridad. Es el caso de: **starting error recovery**, **starting error**, **running error recovery** y **running error**.

La funcionalidad propia de cada componente es implementada en el metaestado **runnig**. Se ha advertido anteriormente que "running" no es realmente un estado, sino que podría definirse como un autómata interno diseñado por el propio usuario para conseguir la funcionalidad deseada para el componente.

9.3.4.4. Intercomunicación entre componentes

De forma similar a la comunicación entre procesos (IPC, Inter Process Communications) aportada por los actuales sistemas operativos, CoolBOT utiliza un sistema de comunicación entre componentes (ICC, Inter Component Comunications). Este modelo estandariza las comunicaciones, permitiendo el trabajo cooperativo entre componentes a la par que manteniéndolos desacoplados, lo que favorece la reutilización y desarrollo independiente de componentes.

El modelo de intercomunicación utilizado en CoolBOT se basa en los puertos de entrada y de salida de los componentes, realizando conexiones entre los mismos. Los datos se transmiten a través de estas conexiones en forma de paquetes de datos de distintos tipos denominados "port packets".

Estos "port packets" son unidades discretas que contienen la información que un componente desea transmitir. A través de la operación de "marshalling" los datos son encapsulados para su envío por la red, de tal forma que represente un formato adecuado para la transmisión. Especialmente cuando ésta se realiza entre componentes situados en diferentes equipos de una red de computadores. Cuando se reciben datos la operación de "demarshalling" extrae los datos del paquete.

Tipo de "port packet"	**Descripción**
PacketUChar	Transporta un tipo *unsigned char* de C++.
PacketInt	Transporta un tipo *int* de C++.
PacketLong	Transporta un tipo *long* de C++.
PacketDouble	Transporta un tipo *double* de C++.
PacketTime	Transporta un tipo *Time* de CoolBOT (instante de tiempo).
PacketCoordinates2D	Transporta un tipo *Coordinates2D* de CoolBOT (almacena un punto 2D).
PacketCoordinates3D	Transporta un tipo *Coordinates3D* de CoolBOT (almacena un punto 3D).
PacketRGBImage	Transporta una imagen del tipo RGB (Red, Green y Blue).

Tabla 9.2: Algunos de los tipos de "port packets" definidos por defecto en CoolBOT.

Existen diferentes tipos de "port packets" ya definidos en CoolBOT como los indicados en la tabla 9.2. No obstante, si se desea transmitir un tipo de dato que aún no dispone de su propio "port packet" CoolBOT proporciona las herramientas para la creación de estos paquetes por parte del usuario.

9.3.4.5. Conexiones, puertos y paquetes de puerto

Los componentes en CoolBOT se comunican unos con otros utilizando conexiones entre puertos. Éstas se definen por una conexión entre un puerto de salida y un puerto de entrada. Dichas conexiones son unidireccionales, van desde un puerto de salida a uno de entrada, y son establecidas en tiempo de ejecución.

Disponen de ciertas características como son:

- Están desacopladas, en el sentido de que el componente que publica datos no conoce necesariamente quién recibe esos datos y el componente que recibe datos no conoce necesariamente al emisor de los mismos.
- Siguen el paradigma de "publicación/subscripción", por el cual un componente que publica datos puede tener múltiples suscriptores que los reciben, o bien un componente puede recibir datos publicados por múltiples componentes. En las figuras 9.5 y 9.6 incluidas a continuación se pueden observar estas posibilidades.

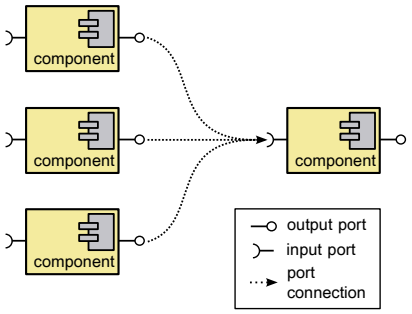

Figura 9.5: Múltiples componentes emisores y sólo uno receptor.

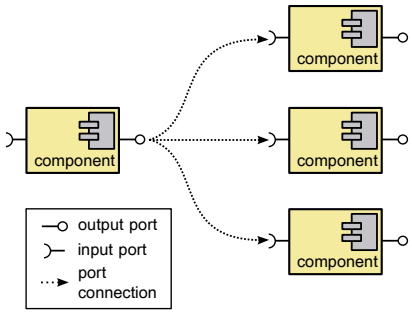

Figura 9.6: Un único emisor y múltiples receptores.

Para establecer este tipo de conexiones ambos puertos (entrada y salida) han de ser compatibles, entendiendo por compatible que tienen tipos de datos compatibles y que deberán transportar el mismo tipo de "port packet". De forma particular, cuando se define un puerto de salida o entrada es indispensable especificar tres aspectos:

1. **Identificador** o nombre de puerto, debe ser único para el componente. Este nombre será al cual nos referiremos al establecer o eliminar conexiones entre puertos.
2. **Tipo de puerto**. Hay una variedad de tipologías disponibles para puertos de entrada y salida, dependiendo de cómo se establezcan podremos establecer diferentes modelos de comunicación para cada conexión.

3. **Tipo de paquete de puerto**. Son los tipos de paquetes, "port packets", aceptados por los puertos de entrada o salida y en donde se almacena la información enviada o recibida. La mayoría de puertos sólo admiten un tipo de "port packet" a través de ellos.

Es importante recordar que aunque las conexiones entre puertos se establecen dinámicamente, la definición de cada puerto es estática dentro de cada componente. Entendiendo por definición de un puerto la asignación de un identificador, tipo de puerto y tipo "port packet" que acepta.

9.3.4.6. Vistas

CoolBOT está orientado al diseño de componentes y su interconexión para obtener funcionalidades definidas. No obstante, en ocasiones también se requiere mostrar datos o información específica. Con ese fin existen las denominadas **vistas**, figura 9.7.

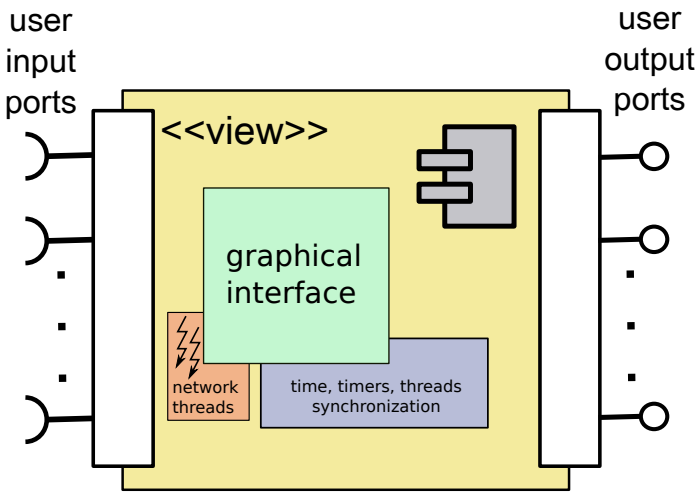

Figura 9.7: Esquema de una vista en CoolBOT

Las *vistas* de CoolBOT son interfaces gráficas (GUI) desacoplables, integrables y reutilizables disponibles para sistemas desarrollados e integrados en CoolBOT. Estas vistas, como componentes software que son, pueden interconectarse con cualquier otro tipo de componente en un sistema CoolBOT.

En la figura 9.7 podemos ver la estructura de una vista en CoolBOT. Como se observa, están provistas de una interfaz externa con puertos de entrada y salida. De hecho, las vistas utilizan la misma infraestructura proporcionada por el framework para comunicarse con componentes, vistas o sondas en un sistema, ya residan todas en una misma integración o de forma remota en otra.

Al igual que los *componentes*, internamente las vistas disponen del mismo soporte de red el cual permite comunicaciones de red transparentes y desacopladas a través de conexiones entre puertos. Al contrario que los componentes, las

vistas no están diseñadas para realizar ningún tipo de funcionalidad, más allá de la implementación de una interfaz gráfica que sirva como salida visual para los datos.

Debido a su mayor simplicidad las vistas no hacen uso de un autómata por defecto para modelar su funcionamiento, al contrario que en los componentes CoolBOT. Para la creación de la interfaz gráfica CoolBOT utiliza la librería gráfica GTK+[13].

9.3.4.7. Integraciones

Una **integración** es una composición de *componentes*, *vistas* y *sondas*. Estas integraciones son definidas por el usuario para conseguir una funcionalidad en concreto con el sistema CoolBOT creado.

La figura 9.8 muestra un ejemplo de integración. Concretamente se trata de una integración distribuida, compuesta por multitud de componentes y vistas, así como una sonda, situadas en diferentes máquinas.

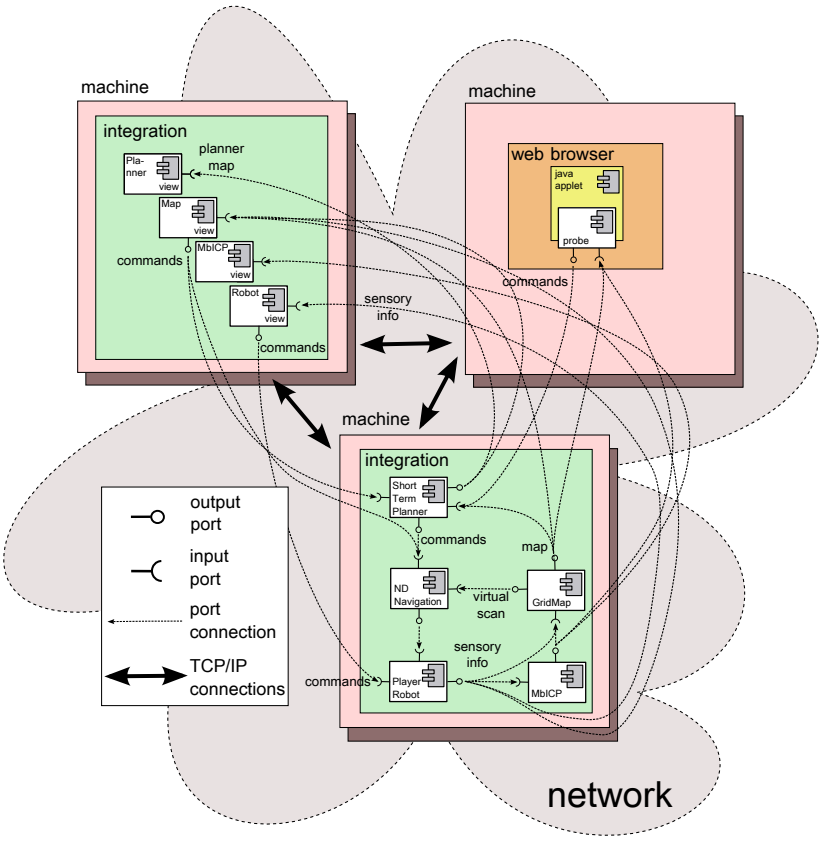

Figura 9.8: Ejemplo de integración en CoolBOT

9.3.4.8. Herramientas de desarrollo

Como parte de su estructura, CoolBOT proporciona varias herramientas para ayudar al usuario a la hora de desarrollar sistemas. Las principales son *coolbot-*

bundle y *coolbot-c*, como se puede ver en la figura 9.9.

La primera de ellas, **coolbot-bundle**, es utilizada para crear los denominados "**bundles**". Un "bundle" es un espacio de trabajo, con estructura de directorio, para el desarrollo de software en CoolBOT el cual puede contener componentes, sondas, vistas, grupos de "port packets" (unidades discretas de información que intercambian los componentes CoolBOT) o integraciones. Los "bundles" disponen de una infraestructura CMake[4] para la configuración y compilación de todo el software CoolBOT incluido en él, de tal forma que todo lo que está incluido en el "bundle" es compilado de forma integrada. Además, una vez finalizado el desarrollo del contenido, el "bundle" también proporciona la infraestructura CMake para su instalación.

De forma general, cuando se inicia un proyecto se crea un nuevo "bundle" dentro del cual se definen todos los componentes, sondas, vistas, "port packets" e integraciones relacionados con dicho proyecto. Esto facilita el desarrollo a lo largo del proyecto, la instalación del software una vez terminado y la localización de esos recursos.

La otra herramienta, **coolbot-c**, es el compilador de CoolBOT. Éste genera esqueletos en C++ para componentes, "port packets", vistas e integraciones, y por cada componente también genera su correspondiente sonda. Todos estos esqueletos son clases en C++. Para producir dichas clases *coolbot-c* utiliza un lenguaje descriptivo como código fuente, también llamado código CoolBOT. Las clases generadas están incompletas (excepto las sondas), de ahí que se las defina como esqueletos, ya que les falta la funcionalidad que el usuario será encargado de aportar. Una vez completadas, estas clases pueden ser compiladas con los templates de CMake aportados por la herramienta *coolbot-bundle*.

Al compilar componentes, sondas, vistas o "port packets" se generan librerías dinámicas, mientras que al compilar integraciones se generan programas ejecutables. Por otra parte, el compilador *coolbot-c* conserva la información de los archivos de descripción modificados cuando se recompila, de tal manera que todo el código C++ introducido por el usuario en los esqueletos se mantiene intacto. El uso de *coolbot-c* queda algo más claro en la figura 9.9.

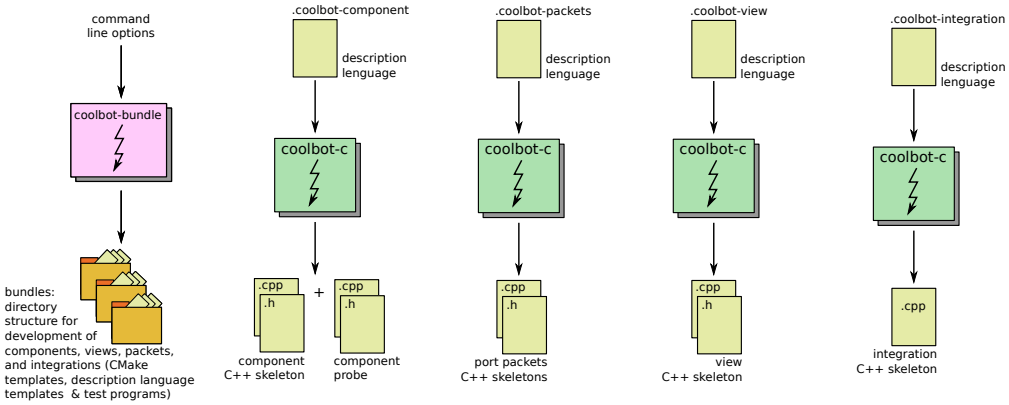

Figura 9.9: Proceso de desarrollo software en CoolBOT

Capítulo 10

Diseño e Implementación

En este capítulo se explicará el software desarrollado a lo largo del Trabajo Fin de Grado, así como la naturaleza de su diseño.

El código desarrollado a lo largo de esta etapa es a nuestro entender ampliable y flexible, de tal forma que cualquier desarrollador que desee añadir alguna funcionalidad adicional a las implementadas en este proyecto pueda hacerlo. A su vez, el código implementado mantiene una organización modular de manera que resulte sencillo y limpio.

También se ha dotado de una interfaz amigable para los usuarios que utilicen las herramientas generadas, especialmente en el sistema de gestión de vídeo distribuido, a fin de que resulten cómodas e intuitivas.

Casi en su totalidad el trabajo ha sido desarrollado utilizando el framework de programación CoolBOT[6], exceptuando las fases iniciales. Ello añade al proyecto una serie de características implícitas a CoolBOT como: mayor modularidad, control de errores, sistema distribuido, simplicidad de diseño, entre muchas otras.

Siguiendo los objetivos marcados para el trabajo, se ha divido esta fase de diseño e implementación en tres subapartados diferentes:

1. **Integración de cámaras IP**: fase centrada en la creación de una infraestructura que permita una integración de cámaras IP Axis, utilizando para ello el marco de programación CoolBOT[6].

2. **Almacenamiento en vídeo**: etapa centrada en la creación de vídeo a partir imágenes obtenidas de cámaras IP, utilizando para ello una biblioteca de tratamiento de vídeo.

3. **Sistema de vídeo distribuido**: creación de un sistema de vídeo distribuido que permita el acceso simultáneo a múltiples vídeos grabados en la etapa anterior.

Cada etapa se desarrolla en base a la anterior, siguiendo así un modelo de trabajo incremental.

10.1. Integración de cámaras IP

El principal objetivo del presente Trabajo Fin de Grado es integrar **cámaras IP**, dispositivos de vídeo capaces de transmitir imágenes a través de la red, en el framework de programación CoolBOT[6], marco de programación orientado a componentes para sistemas robóticos diseñado y desarrollado en el Instituto Universitario SIANI y en el DIS. De esta forma, se creará una **infraestructura software** que proporcione al usuario un acceso integrado online a cámaras IP.

En el capítulo anterior de "Análisis del problema", sección 9, se ha podido ver como para la interacción con las cámaras IP se utiliza la propia API de estos dispositivos, llamada **VAPIX**[36]. Asimismo se hará uso de la biblioteca **libcurl**[21] para el envío y recepción de datos a través de la red. Ambas herramientas han sido estudiadas previamente en las secciones 9.3.1 (VAPIX) y 9.3.2 (libcurl).

En el estudio de VAPIX, sección 9.3.1, se puede observar el **formato** con el que envía la cámara IP el **flujo de imágenes** para el comando seleccionado. Dicho formato está compuesto por una cabecera propia al inicio de cada imagen y los datos de la misma. La figura 10.1 muestra una representación gráfica del formato en el que VAPIX proporciona las imágenes, mientras que en la figura 10.2 se puede observar un ejemplo genérico de dicho formato.

Otra característica destacada de VAPIX es que los datos son enviados por la cámara IP por partes, es decir, **los datos de cada imagen se encuentran troceados**. Tanto el formato de los datos como la manera en la que se envían son aspectos muy importantes a tener en cuenta para diseñar y desarrollar una correcta integración.

Una vez destacados estos aspectos, se plantea la integración de cámaras IP gradualmente. En primer lugar se realiza una **fase inicial** en la que establecer comunicación entre ordenador y cámara IP. Posteriormente se realiza un **prototipo inicial**, no integrado en CoolBOT, que sirva de primera aproximación a la integración de una cámara de red. Finalmente el prototipo se adaptará a CoolBOT[6] para obtener una **infraestructura software** capaz de proporcionar un acceso integrado online a las imágenes de la cámara de red.

10.1.1. Fase inicial

La fase inicial de desarrollo supone una primera aproximación para establecer una comunicación entre el ordenador y la cámara IP. En esta etapa se elaboran diversos programas previos al prototipo inicial con los que aprender el correcto uso de las herramientas libcurl[21] para enviar y recibir datos al dispositivo a través del protocolo de red HTTP[14] (Hypertext Transfer Protocol).

Figura 10.1: Representación gráfica del formato

```
--myboundary
Content-Type: image/jpeg
Content-Length: 15656


<JPEG image data>


--myboundary
Content-Type: image/jpeg
Content-Length: 14978


<JPEG image data>


--myboundary
Content-Type: image/jpeg
Content-Length: 15136


<JPEG image data>


--myboundary
Content-Type: image/jpeg
Content-Length: 15012

...
```

Figura 10.2: Formato de envío de datos

10.1.1.1. Pre-prototipo 0

Este pre-prototipo proporciona una primera toma de contacto con la biblioteca libcurl[21] estudiada en la sección 9.3.2. Con esta herramienta se solicitará un flujo de imágenes a la cámara IP utilizando para ello un comando de la API VAPIX[36] vía HTTP[14].

En este caso se utiliza la "**easy interface**", una interfaz de libcurl que proporciona una serie de funciones para transferencias bajo los protocolos soportados de forma síncrona, eficiente y de fácil uso.

Inicialmente el pre-prototipo inicia una sesión de esta interfaz "easy" y establece las opciones para la transferencia a realizar. Las opciones concretas son:

1. **URL**[35] para localizar el recurso en red y realizar la petición. Esta URL corresponde al comando VAPIX para solicitar un "stream" o flujo de imágenes a la cámara de red. El dispositivo es localizado correctamente al estar la URL compuesta por su dirección IP, entre otros parámetros.
2. **Método de autentificación**. Se establece el método de acceso **digest**[7] que aplica una función de hash[10] a la contraseña antes de ser enviada por la red, de tal modo que proporcione cierta seguridad.
3. **Usuario** de autentificación en el dispositivo.

4. **Contraseña** de autentificación en el dispositivo.

Posteriormente se solicita el flujo de imágenes a la cámara de red mediante la función libcurl correspondiente. Por defecto, los datos recibidos por parte de la cámara IP se mostrarán por pantalla hasta que el emisor finalice la transferencia. En la figura 10.3 se representa su funcionamiento.

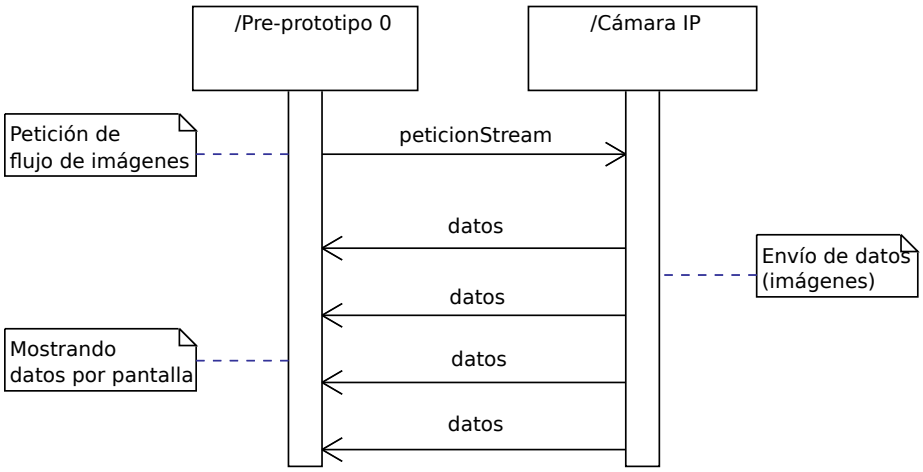

Figura 10.3: Transferencia entre pre-prototipo 0 y cámara IP

Por último, sólo resta finalizar la sesión libcurl.

10.1.1.2. Pre-prototipo 1

Con el pre-prototipo 0 se ha logrado una primera comunicación con la cámara IP. Sin embargo, el **tratamiento de los datos** y la **finalización de la transferencia** son aspectos a modificar. Por ello, se crea un nuevo pre-prototipo que solucione estos elementos conflictivos.

La "easy interface" de libcurl permite transferencias síncronas y eficientes, no obstante, no permite al receptor **finalizar la comunicación**. Esto constituye un problema ya que la cámara IP genera imágenes continuamente y no finaliza en ningún momento la conexión. Por este motivo el pre-prototipo 1 utiliza la "**multi interface**" de libcurl, la cual proporciona funciones que posibilitan transferencias múltiples y/o asíncronas.

Otro aspecto a mejorar es el **tratamiento de los datos**. En el pre-prototipo 0, sección 10.1.1.1, los datos son simplemente mostrados por pantalla. En su lugar se definirá una función propia que reciba los datos y pueda proceder a su tratamiento, para ello se ha de especificar en las opciones de la transferencia. Con estos cambios el programa queda de la siguiente manera.

En primer lugar se inicia la sesión de la interfaz "multi" y la sesión de la interfaz "easy", observe que la "multi interface" se compone de "easy interfaces". Se establecen las opciones de transferencia, idénticas al pre-prototipo 0 con la

excepción de definir la función que reciba los datos para ser tratados.

A continuación se inicia la transferencia de datos solicitando las imágenes a la cámara IP. En esta ocasión los datos recibidos en el pre-prototipo 1 serán suministrados a la función indicada en las opciones de transferencia. La transferencia queda así como en la figura 10.4.

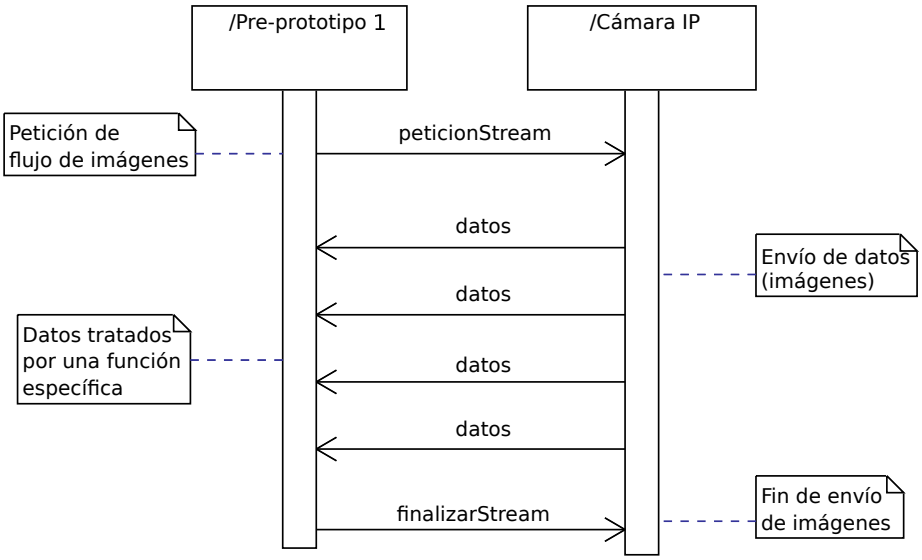

Figura 10.4: Transferencia entre pre-prototipo 1 y cámara IP

Gracias a la "multi interface" el pre-prototipo 1 es capaz de finalizar la transferencia cuando lo desee. Finalmente se cierran las sesiones abiertas de ambas interfaces.

Con este pre-prototipo se consigue comunicar ordenador y cámara IP haciendo uso de las herramientas libcurl. El siguiente paso es el desarrollo de un prototipo que principalmente trate la información y se beneficie de lo implementado en esta fase inicial.

10.1.2. Prototipo inicial

Este prototipo inicial tiene como objetivo integrar una cámara IP, abstrayendo al usuario de los detalles internos de la comunicación.

El presente prototipo representa una librería compuesta por numerosas funciones que facilitan la conexión con el dispositivo, autentificación en el mismo y el tratamiento de la información recibida a través de la red.

El desarrollo del prototipo se realiza de manera incremental, para lo cual se diseñan una serie de etapas que amplíen de manera progresiva las funcionalidades. En las diferentes etapas se crean diversas funciones para lograr la integración de cámaras IP.

Las etapas diseñadas son las siguientes.

- **Etapa 1**. En la primera etapa se genera una conexión inicial con la cámara IP que permita identificar posibles errores de la conexión como autentificación errónea, dispositivo no encontrado, etc. Por este motivo se procederá a la creación de una función que establezca dicha conexión y detecte posibles errores.
- **Etapa 2**. Tras comprobar que la conexión es correcta se solicita a la cámara IP un "stream" o flujo de imágenes. Para ello se utilizará el comando VAPIX[36] correspondiente y el software libcurl[21] para su transmisión. Se tomará como referencia para esta etapa el pre-prototipo 1 creado en la fase inicial, sección 10.1.1.
- **Etapa 3**. Creación de una función capaz de detectar una ristra de caracteres específica dentro de un bloque de datos. Esta función será necesaria para el posterior tratamiento de los datos enviados por la cámara IP.
- **Etapa 4**. Creación de un nuevo tipo de buffer capaz de almacenar los datos de una imagen y realizar un manejo eficiente de la memoria dinámica. El buffer será de gran utilidad en el posterior tratamiento de los datos enviados por la cámara de red.
- **Etapa 5**. Una vez solicitado el flujo de imágenes se requiere tratar la información recibida del dispositivo de red. Con dicha finalidad se procede a la creación de una función que reciba dichos datos y que extraiga cada imagen contenida en ellos, para lo cual utilizarán las funciones y el buffer de las etapas 3 y 4, respectivamente. Esta nueva función ha de especificarse entre las opciones de la transferencia, etapa 2, para poder recibir los datos de la cámara IP.

Las funciones implementadas a lo largo de las etapas componen el prototipo inicial. Así el programa quedaría estructurado de la siguiente manera.

- **Prueba de conexión**. Función que realiza una primera conexión que compruebe posibles errores de conexión.
- **Extractor de trama**. Función que analice los bloques de datos para almacenar las imágenes. A su vez esta función está dividida en dos nuevas funciones.
  - **Detectar inicio**. Función que encuentre el inicio de la imagen en un bloque de datos.
  - **Buffer propio**. Una clase que implemente un nuevo buffer para almacenar las diferentes partes de la imagen en memoria.

Cabe destacar que la petición del flujo de imágenes a la cámara IP, indicada en la etapa 2, no dispone de una función propia en el prototipo y, por lo tanto, se realiza como en la fase inicial 10.1.1.

A continuación se explican las funciones que componen el prototipo. Cada una de ellas tiene en consideración diferentes características en base a su cometido como, por ejemplo, el formato en que VAPIX suministra el flujo de imágenes en la función encargada de extraer las imágenes.

10.1.2.1. Prueba de conexión

La función **testConnection** se encarga de establecer una conexión inicial con la cámara IP. Su objetivo es comprobar que se cumplen las condiciones idóneas para posteriormente poder interactuar con el dispositivo.

Como parámetros necesita la dirección IP o nombre de dominio de la cámara de red y los los datos de autentificación. Una vez suministrados realiza la petición de conexión, enviando un comando de VAPIX[36] al dispositivo a través del protocolo HTTP, mediante libcurl[21].

La cámara IP responde a la conexión con una serie de datos, entre ellos el código de estado HTTP. Comprobando este código la función determina si la conexión fue exitosa o si ocurrió algún tipo de fallo específico.

Los posibles errores de conexión que comprueba esta función son los siguientes.

- Conexión errónea. La dirección IP o dominio de red especificado de la cámara de red es erróneo.
- Sin contenido. Petición recogida pero no hay datos.
- Petición errónea. La petición no se ha podido realizar al haberse realizado de forma errónea.
- Sin autorización o autentificación errónea. No se tiene permisos para acceder a la cámara IP o los datos de autentificación no son correctos.
- No encontrado. No se ha encontrado nada que coincida con la petición.
- Error interno. Error interno del dispositivo de red.

De manera adicional durante la conexión también realiza la autentificación en el dispositivo, por tanto, si todo ha ido bien, la cámara IP estará preparada para nuevas peticiones. La figura 10.5 muestra el funcionamiento de la función.

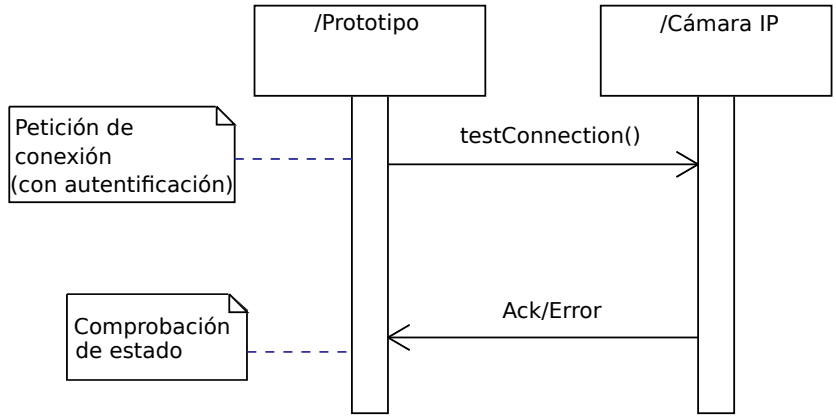

Figura 10.5: Prueba de conexión

10.1.2.2. Detectar inicio

Esta función, denominada **myMemmem**, se encarga de buscar una ristra de forma "inteligente" en un bloque de datos. Su funcionalidad es muy similar a la función "`strstr`" de la biblioteca string.h, salvo por el hecho de que "`strstr`" realiza la búsqueda en un string y no en un bloque de memoria.

Con anterioridad se ha comentado que VAPIX[36] envía los datos de las imágenes por trozos a través de la red. La presente función facilita la detección del inicio de cada imagen aprovechando para ello el formato con el que las cámaras de la compañía Axis envían el flujo de imágenes. Dicho formato es visible en la figura 10.2 y se compone de una cabecera (`--myboundary`) al inicio de los datos de cada imagen.

10.1.2.3. GrowingBuffer

GrowingBuffer es un nuevo tipo de buffer. Cuando se introducen nuevos datos en él aumenta su espacio, de forma dinámica, si el espacio del que dispone no es suficiente para almacenarlos.

Cuando se quieren guardar datos nuevos, el buffer los almacena al final de los ya existentes. La extracción de datos se puede realizar en cualquier momento, no obstante, el espacio reservado en memoria se mantiene. El destructor de la clase será el encargado de liberar el espacio en memoria.

La implementación de este buffer se ha realizado con una clase en C++ que dispone de las siguientes funciones públicas:

- *init*. Función que devuelve el inicio del buffer para la extracción los datos.
- *size*. Indica la cantidad de datos almacenados en el buffer.
- *put*. Función encargada de añadir nuevos datos al final del buffer, es necesario especificar cuánto ocupan.
- *newFrame*. Vacía el GrowingBuffer. Función utilizada para desechar los datos o certificar que el buffer está vacío.

Internamente existen otras dos funciones privadas, no accesibles por el usuario.

- *resize*. Comprueba en base a variables internas si es necesario ampliar el buffer para hacer espacio a nuevos datos. Es utilizada por la función "put".
- *realloc*. Amplia el espacio en memoria dinámica del buffer, manteniendo los datos ya alojados.

El diagrama de clase de la figura 10.6 muestra en mayor detalle la clase GrowingBuffer.

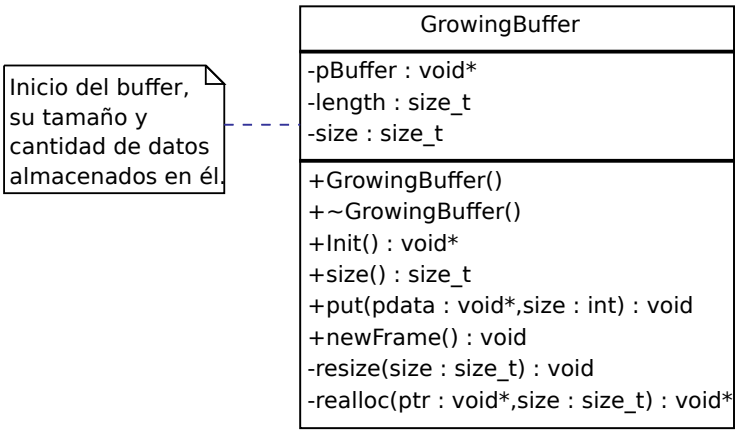

Figura 10.6: Diagrama de la clase GrowingBuffer

En el prototipo, la clase GrowingBuffer es utilizada para **almacenar los diferentes trozos de una imagen**, de tal forma que una vez obtenidos todos los trozos el buffer dispone de los datos completos de una imagen. Al extraer los datos de una imagen se aprovecha el espacio de memoria ya reservado para la siguiente. Gracias a su capacidad de crecer, a la larga el GrowingBuffer se adaptará al tamaño de la imagen que más ocupe y, por lo tanto, no necesitará ampliarse más.

Con este buffer se consigue un manejo eficiente de la memoria. La figura 10.7 muestra gráficamente el funcionamiento de un GrowingBuffer. Los elementos t0, t1, t2,.. representan diferentes instantes de tiempo.

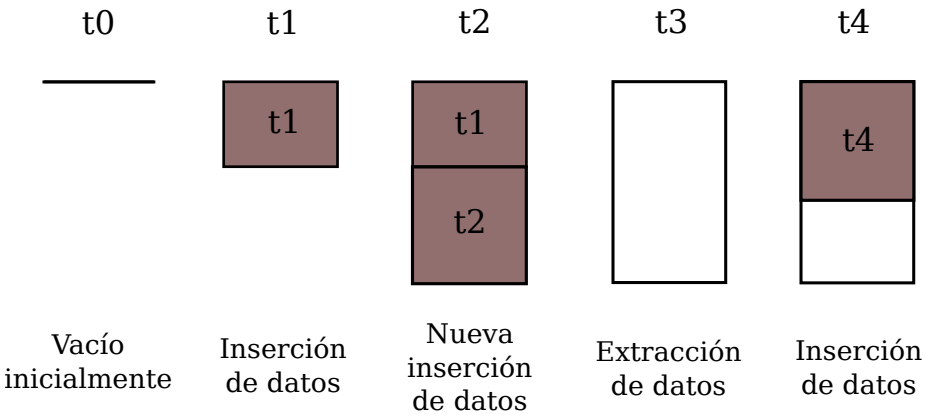

Figura 10.7: Funcionamiento GrowingBuffer

Se puede observar como inicialmente el buffer se encuentra vacío. En los instantes de tiempo t1 y t2 se insertan datos, como consecuencia en ambas ocasiones se reserva memoria para almacenarlos. Los datos son guardados al final de los ya existentes en cada caso. En el instante de tiempo t3 se extrae el contenido del GrowingBuffer, sin embargo, el espacio reservado en memoria se mantiene para ser aprovechado en futuros instantes de tiempo, como en t4.

10.1.2.4. Extractor de trama

A lo largo del análisis de libcurl[21], sección 9.3.2, y la fase inicial, sección 10.1.1, se ha visto que es posible definir una función propia que reciba los datos de la transferencia para su tratamiento. De esta manera, dicha función será ejecutada cada vez sean recibidos nuevos datos.

El prototipo aprovecha esta funcionalidad creando una función que reciba la información procedente de la cámara IP para tratarla adecuadamente. La figura 10.8 muestra cómo los datos son recibidos por esta función, denominada **frameParser**.

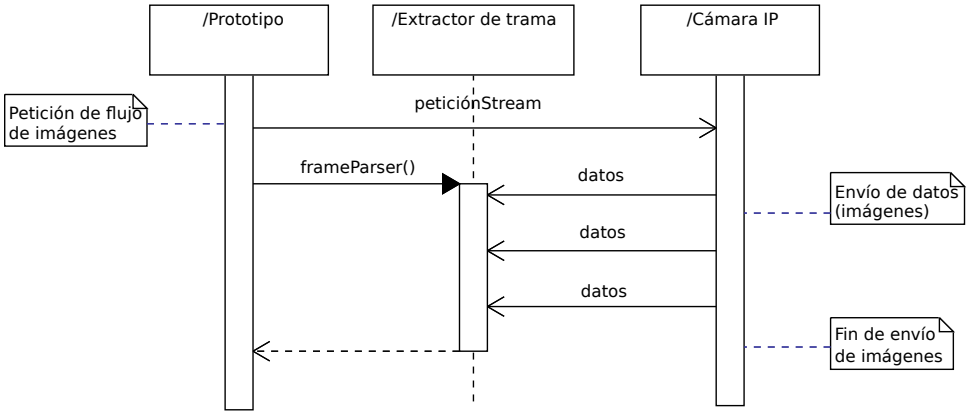

Figura 10.8: Conexión flujo de imágenes

Para el correcto diseño de la función frameParser es necesario saber **cómo son enviados y qué formato poseen los datos** procedentes de la cámara de red. Durante el estudio de libcurl, sección 9.3.1.3, y al inicio de la integración, sección 10.1, se ha especificado que VAPIX[36] envía los datos de las imágenes por trozos, es decir, envía pequeños bloques de datos de forma periódica que conforman el flujo de imágenes. Además, los datos poseen el formato representado en la figura 10.2.

Teniendo en cuenta estos aspectos la función frameParser actúa como un **extractor de trama**. Su cometido es detectar dentro de cada bloque de datos recibido qué datos concretos pertenecen a una imagen, almacenar sus diferentes trozos y, una vez conseguido los datos completos de dicha imagen, guardarla en un archivo de disco.

En su implementación se vale de la función detectora de inicio, myMemmem, y el buffer GrowingBuffer desarrollados en apartados anteriores.

- El limitado tamaño de los bloques de datos recibidos no permite el envío de los datos completos de una imagen. Por tanto, es posible deducir que estos bloques transportan trozos de información relativa a una o dos imágenes, en caso de que se solapen los datos de fin de una imagen e inicio de la siguiente.

Para discernir si los datos pertenecen a una única imagen o hay datos de dos imágenes distintas se utiliza la función **myMemmem**, sección 10.1.2.2.

Esta función buscará en cada bloque de datos recibidos la cabecera añadida por VAPIX al inicio de cada imagen ("`--myboundary`"). En caso de encontrarla significará que los datos superiores a dicha cabecera pertenecen a una imagen y los posteriores a ella pertenecen a otra distinta. En caso de no ser encontrada todos los datos de ese bloque pertenecen a una misma imagen.

- El buffer **GrowingBuffer**, sección 10.1.2.3, será utilizado por el extractor de tramas para guardar los diferentes trozos de una imagen. Una vez completos todos sus datos se extrae la imagen del GrowingBuffer para almacenarla. De esta manera el buffer queda nuevamente a disposición de frameParser para almacenar los datos de una nueva imagen.

Siguiendo estas dos pautas el extractor de tramas queda como en la figura 10.9.

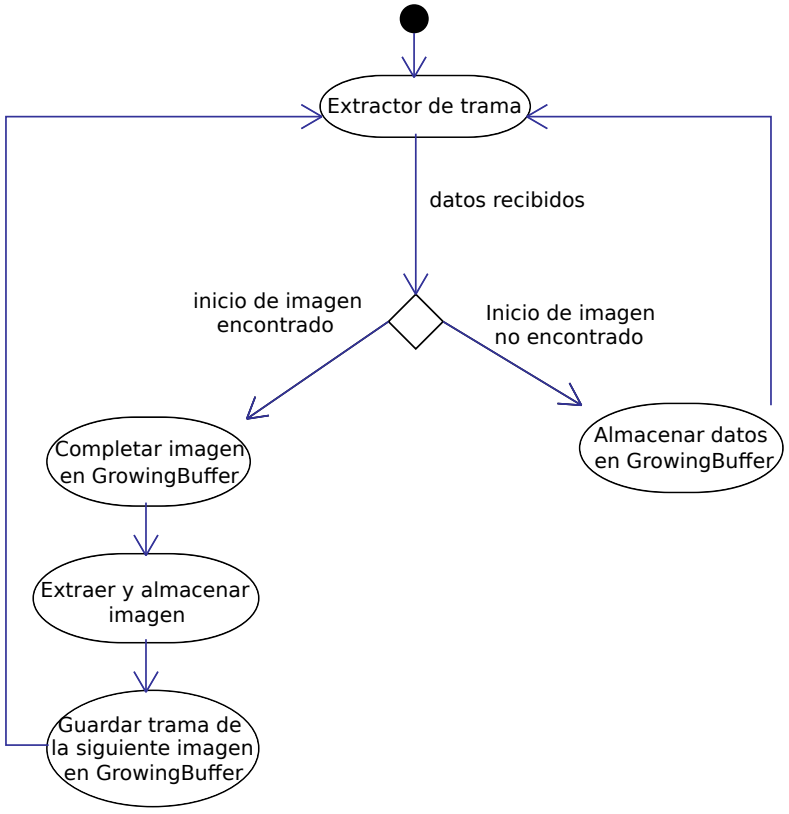

Figura 10.9: Autómata de la función analizadora

De esta forma la función **frameParser** obtiene las imágenes emitidas por la cámara IP en tiempo real hasta que el usuario decida finalizar la ejecución.

Conjugando las diferentes funciones y la clase desarrollada, este primer prototipo ofrece una integración con una cámara IP que abstrae al usuario de múltiples detalles internos a la comunicación y el tratamiento de los datos.

10.1.3. Infraestructura software

Con la culminación del prototipo se dispone de una primera integración de cámaras IP Axis. El siguiente paso es realizar una nueva integración de este tipo de cámaras en CoolBOT[6] adaptando para ello el código a esta plataforma.

Recordemos que CoolBOT[6] es un framework o marco de programación orientado a componentes para sistemas robóticos. Este framework presenta un modelo de programación basado en el concepto de **componente software**, donde los elementos que componen el software de un sistema robótico se denominan componentes.

Los proyectos CoolBOT son elaborados en "**bundles**", espacios de trabajo con estructura de directorio para el desarrollo de software en CoolBOT. Los "bundles" disponen de una infraestructura CMake[4] para configurar y compilar todo el software incluido en él, consiguiendo que todo sea compilado de forma integrada. Una vez finalizado su desarrollo, los "bundles" proporcionan la infraestructura CMake para su instalación.

En el presente TFG se ha creado un nuevo "bundle", llamado **coolbot-camera-ip-bundle**, para implementar todo el software CoolBOT relacionado con *cámaras IP Axis*. Esto facilita el desarrollo a lo largo del proyecto, así como la instalación del software una vez terminado.

Existen numerosos "bundles" ya desarrollados en CoolBOT. Entre ellos cabe destacar **coolbot-camera-bundle** el cual contiene numerosos componentes para la reproducción y grabación de vídeos procedentes de *cámaras web* o *webcams*. Debido a la similitud entre ambos proyectos en ocasiones se utilizarán componentes o vistas de coolbot-camera-bundle en el TFG para aprovechar funcionalidades ya creadas o tomarlas de referencia.

10.1.3.1. Componente CameraIPAxis

Los componentes CoolBOT son unidades que implementan y aportan funcionalidad a un sistema. Para conseguir una integración de cámaras IP Axis se crea un nuevo tipo de componente el cual adapta las funciones creadas en el prototipo inicial, sección 10.1.2. Este componente toma el nombre de **CameraIPAxis** y se encuentra en el "bundle" coolbot-camera-ip-bundle.

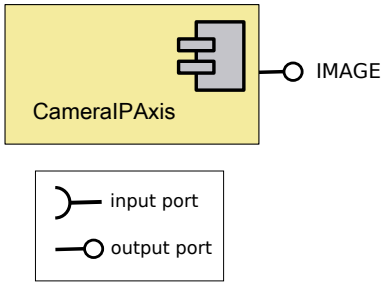

Figura 10.10: Componente CameraIPAxis

El componente **CameraIPAxis** posee un entramado lógico que implementará un sistema multihilo con un único puerto de salida, visible en la figura 10.10. Una vez finalizado este componente transmitirá por su puerto de salida las imágenes generadas por la cámara IP de tal forma que otros componentes o vistas hagan uso de ellas.

A continuación se muestra el código CoolBOT a partir del cual se genera el componente. Se pueden observar parámetros como los frames por segundo (FPS) para la fluidez del vídeo, un estado "Main" con una iteración controlada por un timer y tipos de excepciones no estándar, entre otros parámetros.

Código CoolBOT que representa al componente CameraIPAxis

```
/*
 * File: camera-ip-axis.coolbot-component
 * Description: description file for CameraIPAxis component
 * Date: 05 July 2012
 * Generated by coolbot-bundle version 1.1.0
 */
component CameraIPAxis
{
  header
  {
    author "Bycor Sanchez Sanchez <bycor.sanchez@gmail.com>";
    description "CameraIPAxis CoolBOT component";
    institution "IUSIANI - Universidad de Las Palmas de Gran Canaria";
    version "0.1"
  };

  constants
  {
    public DEFAULT_FPS=25;
    private TIMEOUT_USEC=50000;
  };
  // output port
  output port IMAGE type poster port packet PacketRGBImage;

  exception CONNECTION_ERROR
  {
    description "ip camera connection failed.";
  };
  exception DEVICE_NOT_FOUND
  {
    description "device not found.";
  };
  exception DEVICE_ERROR
  {
    description "device error.";
  };

  entry state Main
  {
    transition on TIMER;
  };
};
```

El principal aspecto a tener en cuenta durante el diseño y desarrollo del componente es que debe **garantizar una frecuencia mínima de envío de imágenes** (Frames Per Second, FPS) para conseguir una visualización fluida de vídeo.

Como se ha visto los componentes son modelados a partir de un autómata por defecto que aporta CoolBOT. En la figura 10.11 es posible observar dicho autómata, el cual consta de múltiples estados y transiciones que corresponden a eventos disponibles por defecto.

Cabe destacar el metaestado **running**, representado en la figura 10.11 con un círculo en raya discontinua. Este metaestado será sustituido por un **autómata diseñado por el usuario** que aporte la funcionalidad concreta al componente.

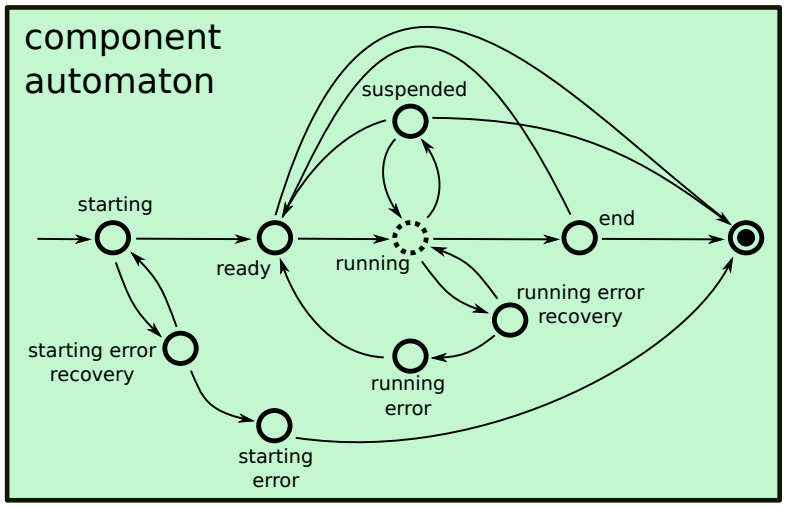

Figura 10.11: Autómata por defecto en componentes CoolBOT

Para adaptar el prototipo inicial, implementado en la sección 10.1.2, al componente CameraIPAxis es necesario plantear las acciones a realizar en los estados del autómata. De este modo se propone el siguiente esquema.

- Realizar la **prueba de conexión**, función "`testConnection`" de la sección 10.1.2.1, en el estado "starting". De este modo si existe algún tipo de fallo de conexión el autómata pasará al estado "starting error recovery" para controlar el error producido. Será necesario modificar la función "`testConnection`" para que genere excepciones en caso de existir errores.
- **Inicializar parámetros** de funcionamiento y operación en el estado "starting". Por ejemplo, los relacionados con las opciones de transferencia en libcurl[21].
- Realizar la **petición de flujo de imágenes** en el estado "ready" para que la cámara IP inicie el envío de imágenes.
- **Ejecución de un autómata propio** en el metaestado "running" que reciba los datos procedentes de la cámara IP, obtenga de ellos cada imagen y envíe por la salida del componente dichas imágenes con una frecuencia mínima. El diseño de este autómata se trata más adelante.
- **Eliminación de recursos** adquiridos durante la ejecución en el estado "end".

El **diseño del autómata** que envíe imágenes de la cámara IP requiere de nuevos mecanismos que garanticen la frecuencia de envío. Esto se debe a que la función "`frameParser`", encargada de la recepción y extracción de imágenes, no

es capaz de garantizar una frecuencia concreta ya que depende directamente de la frecuencia configurada en la cámara de red y del posible retardo entre el envío y la recepción de los datos.

Este aspecto esencial provoca que el autómata diseñado utilice **dos hilos de ejecución** y un método que permita la comunicación entre ambos hilos, en este caso particular la técnica de buffer múltiple. De este modo el autómata que define la funcionalidad del componente CameraIPAxis dispone de las siguientes características.

1. **Hilo libcurl**. Este hilo de ejecución se ejecuta en paralelo o concurrentemente al hilo principal. Se encarga de recibir los datos de la cámara IP y extraer cada una de las imágenes, utilizando para ello la función "`frameParser`" desarrollada en la fase inicial, sección 10.1.2.4.

2. **Hilo principal**. Este hilo de ejecución está compuesto de un estado llamado "Main". Su función es enviar por el puerto de salida del componente la última imagen disponible de la cámara IP. Dispone de una transición asociada a un timer que permite su ejecución iterativa cada cierto periodo de tiempo. Este periodo se configura para mantener una frecuencia de envío de imágenes determinado (FPS).

3. **Técnica de buffer múltiple**. Esta técnica consiste en el uso de más de un buffer para el almacenamiento de un bloque de datos. Cada uno de los hilos dispondrá de un buffer propio, de tal manera que cuando el hilo libcurl disponga de una imagen actualice su buffer y lo intercambie con el buffer del hilo principal. De este modo el hilo principal siempre dispondrá de la última imagen recibida aunque no esté actualizada.

Siguiendo este esquema queda un autómata como el de la figura 10.12. Ambos hilos son ejecutados de manera simultánea.

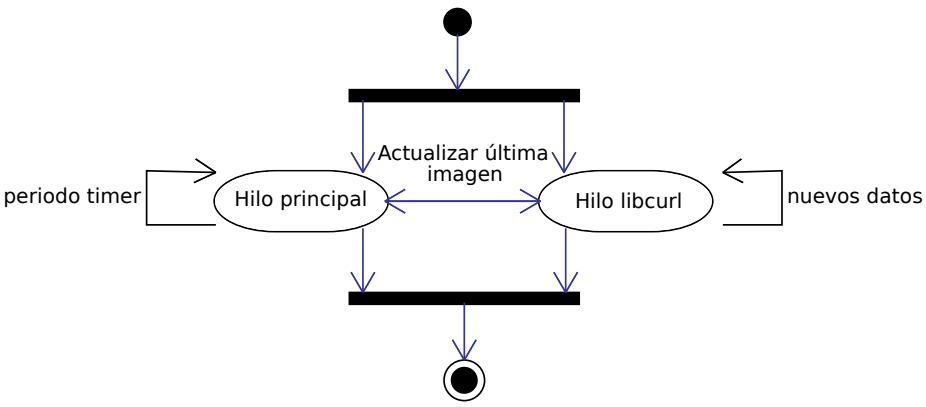

Figura 10.12: Autómata CameraIPAxis

A continuación se explicará con mayor detalle el funcionamiento de estos tres aspectos durante la ejecución del autómata.

Hilo libcurl

Hilo de ejecución creado internamente en el componente CameraIPAxis, motivo por el cual no aparece en su código CoolBOT. Se crea en el estado "starting" del autómata por defecto, para prevenir posibles errores durante su creación. Es lanzado a ejecución en el estado "ready" y eliminado en el estado "end".

En este hilo se adapta el extractor de trama desarrollado en el prototipo, función "**frameParser**", para analizar los datos recibidos de la cámara IP y extraer sus imágenes. Son necesarias una serie de modificaciones en el extractor de trama para adaptar su funcionamiento al componente. En concreto las modificaciones son las siguientes.

- Cada **imagen obtenida es transformada de formato** JPEG[18], formato en el que son enviadas por la cámara IP, a formato RGB (red, green, blue). La transformación se realiza utilizando la función "`imdecode`" perteneciente a la librería HighGui de OpenCV[28].
- Se crea un **buffer adicional**, diferente al GrowingBuffer, que permita suministrar la última imagen disponible al hilo principal mediante la técnica de *buffer múltiple*.
- **La última imagen extraída** en cada momento **es almacenada en el buffer** del paso anterior, en lugar de guardarla en un archivo como ocurría con anterioridad.
- Se realiza la **técnica de buffer múltiple** con el buffer para proporcionar al hilo principal los datos de la última imagen extraída, la sincronización entre ambos hilos se resuelve con el intercambio de punteros protegidos con un cerrojo.

Con estos cambios el autómata de la función "`frameParser`", figura 10.9, se ve modificado en el estado "extraer y almacenar imagen" en el cual ahora ha de: extraer imagen, modificar su formato, almacenar sus datos en el nuevo buffer y realizar intercambio de buffer, técnica de buffer múltiple.

Hilo principal

Hilo de ejecución generado por defecto en CoolBOT. Es el encargado de la ejecución del autómata predefinido, figura 10.11.

Este hilo principal dispone de una **transición asociada a un timer**. Los timers son objetos que facilitan contabilizar el tiempo de una manera precisa. Una transición asociada a un timer permite que un estado se ejecute de forma periódica cada vez que transcurra el tiempo con el que esté configurado.

Concretamente el timer que utiliza el hilo principal se configura para que contabilice los milisegundos que han de pasar entre el envío de una imagen y otra, para ello tiene en cuenta las imágenes por segundo que el usuario desee. Por ejemplo, si se quiere mantener una tasa de 25 imágenes por segundo el timer será configurado con un periodo de 40 milisegundos.

Durante la etapa de ejecución del autómata, figura 10.12, el hilo principal se encarga de **enviar la última imagen disponible** de la cámara IP. Los datos de la última imagen se obtienen del hilo libcurl en un buffer gracias a la técnica de buffer múltiple. El timer garantiza la frecuencia mínima de envío de imágenes para lograr una reproducción fluida.

Las imágenes son enviadas por el puerto `IMAGE` del componente CameraIPAxis, figura 10.10. Los datos de cada imagen son transmitidos a otros componentes en unidades discretas de información, "port packets", en un tipo específico para imágenes en formato RGB llamado `PacketRGBImage`.

Buffer múltiple

El buffer múltiple se define como el uso de más de un buffer para el almacenamiento de un bloque de datos. Si los datos están siendo leídos y escritos al mismo tiempo, el buffer múltiple permite al "lector" obtener una visión completa de los datos (aunque no estén actualizados) en vez de tener una versión parcialmente actualizada de los datos que están siendo creados por el "escritor".

Esta técnica proporciona un **método seguro para actualizar los datos de la última imagen disponible** entre el hilo libcurl y el hilo principal. Los buffers utilizados son del tipo RGBGrid, un tipo de dato proporcionado por CoolBOT para imágenes en RGB (red, green, blue).

Durante el desarrollo de esta técnica existe un **momento crítico** como es el *intercambio entre los buffers* de ambos hilos. Es necesario garantizar que el intercambio se realice de forma atómica para evitar que el hilo principal extraiga datos inválidos. Con este fin se utiliza otro mecanismo de CoolBOT como es un cerrojo (`lock`). Situado donde el hilo principal extrae datos de su buffer y donde el hilo libcurl realiza el intercambio de buffers, el cerrojo evita que ambas acciones se realicen al mismo tiempo.

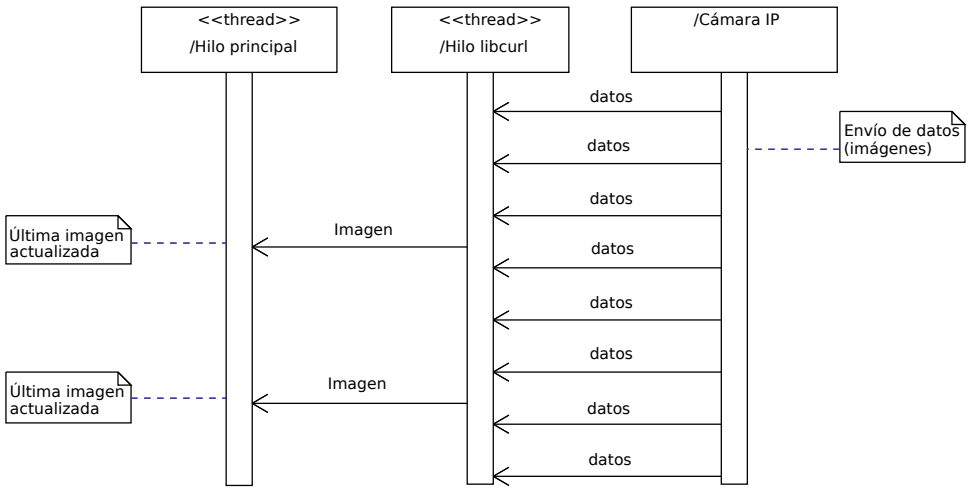

Figura 10.13: Funcionamiento CameraIPAxis

La figura 10.13 muestra la secuencia de funcionamiento entre ambos hilos de ejecución y la cámara IP. En ella se observa como el hilo libcurl va recibiendo los datos de la cámara IP y cada vez que extrae de ellos una imagen la actualiza al hilo principal con la técnica del buffer múltiple.

El hilo principal dispone en todo momento de la última imagen extraída, de tal forma que que es capaz mantener una frecuencia de envío de imágenes concreta.

10.1.3.2. Integración camera-ip-axis-example

El componente CameraIPAxis logra la integración de cámaras IP pertenecientes a la compañía Axis[3] en el marco de programación CoolBOT[6]. Sin embargo, para corroborar su correcto funcionamiento es necesario visualizar las imágenes que transmite.

Se hace necesario el uso de una *vista*, tipo de componente que permite implementar el control gráfico y las interfaces de monitorización en sistemas CoolBOT. Existe una vista simple llamada **CameraView** que permite la reproducción de imágenes. Se encuentra en el "bundle" **coolbot-camera-bundle** dedicado a software CoolBOT relacionado con *cámaras web* o *webcams*.

La vista CameraView posee un puerto de entrada llamado `IMAGE` que admite unidades discretas de información ("port packet") del tipo `PacketRGBImage`, específico para imágenes en formato RGB (red, green, blue).

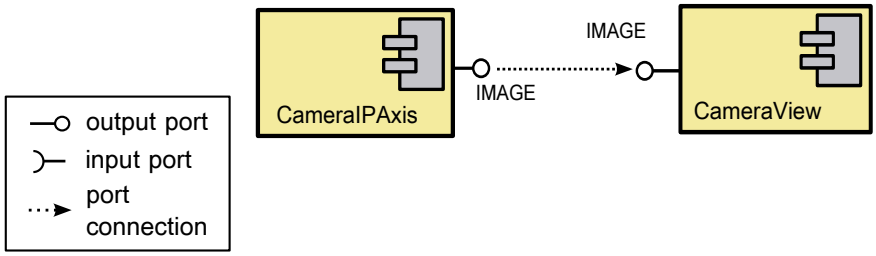

Figura 10.14: Integración camera-ip-axis-example

Conectando el componente CameraIPAxis y la vista CameraView se obtiene una integración que permite visualizar en tiempo real las imágenes de la cámara IP a la que se está conectado. Esta integración, figura 10.14, se denomina **camera-ip-axis-example** y se encuentra en el "bundle" coolbot-camera-ip-bundle.

A continuación se puede ver el código CoolBOT a partir del cual se genera la integración. En él se especifica las instancias del componente y la vista, así como la conexión entre ambos puertos.

Código CoolBOT que representa la integración camera-ip-axis-example

```
/* File: camera-ip-axis-example.coolbot-integration
 * Description: description file for camera-ip-axis-example integration.
 * Date: 27 September 2012
 * Generated by coolbot-bundle version 1.1.0 */
```

```
integration camera-ip-axis-example
{
  header
  {
    author "Bycor Sanchez Sanchez <bycor.sanchez@gmail.com>";
    description "Integration including a CameraIPAxis instance and a
        CameraView instance";
    institution "IUSIANI - Universidad de Las Palmas de Gran Canaria";
    version "0.1"
  };

  machine addresses
  {
    local dis172ac: "dis172acplg.dis.ulpgc.es";
  };
  listening ports
  {
    CAMERA_IP_AXIS_PORT: 1950;
  };

  local instances
  {
    component theCamera:CameraIPAxis listening on CAMERA_IP_AXIS_PORT;
    view theView:CameraView with description "Camera IP Axis View";
  };

  port connections
  {
    connect theCamera:IMAGE to theView:IMAGE;
  };
};
```

Al ser ejecutada, la integración solicita la dirección IP o dominio de red de la cámara IP Axis a la que desea conectarse, usuario y contraseña de autentificación. Ante cualquier problema se especificará por pantalla la posible causa del error.

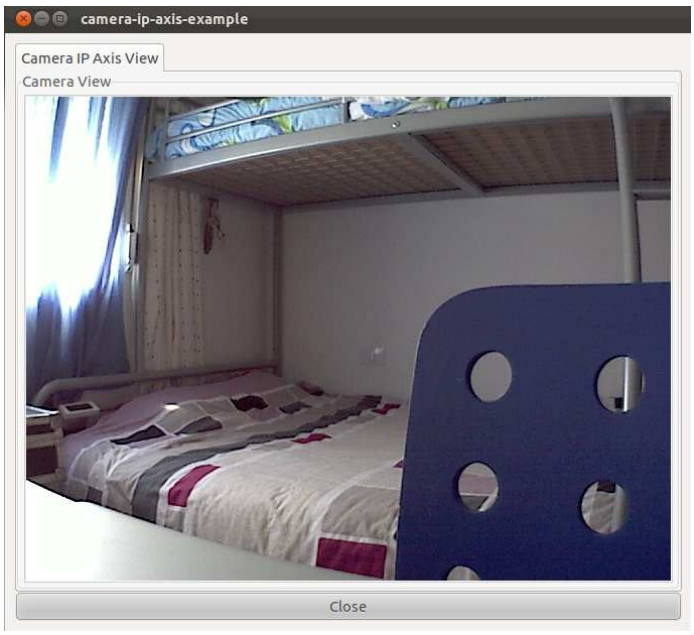

Figura 10.15: Ejemplo de funcionamiento camera-ip-axis-example

La figura 10.15 muestra un ejemplo de su funcionamiento. El botón "close" situado en la parte baja de la pantalla permite finalizar la ejecución.

10.2. Almacenamiento en vídeo

El segundo objetivo marcado en el presente TFG es la creación de archivos de vídeo a partir de imágenes procedentes de cámaras IP Axis.

La idea inicial consistía utilizar la biblioteca de grabación OpenCV[28] estudiada, sección 9.3.3, en un nuevo componente que colaborara con el componente CameraIPAxis y almacenara en un archivo de vídeo las imágenes con la frecuencia a la que son emitidos.

No obstante, en el espacio de trabajo **coolbot-camera-bundle** ya ha sido desarrollado un componente similar haciendo uso de OpenCV. Este "bundle" contiene software CoolBOT relacionado con *cámaras web* o *webcams*. Concretamente el componente se llama **VideoRecorder**.

10.2.1. Componente VideoRecorder

El componente VideoRecorder se encuentra en el **coolbot-camera-bundle**. Dispone de un puerto de entrada llamado `IMAGE` el cual recibe unidades discretas de información, "port packets", tipo `PacketRGBImage`, específico para paquetes que contienen imágenes en formato RGB (red, green, blue). Se puede observar en la figura 10.16.

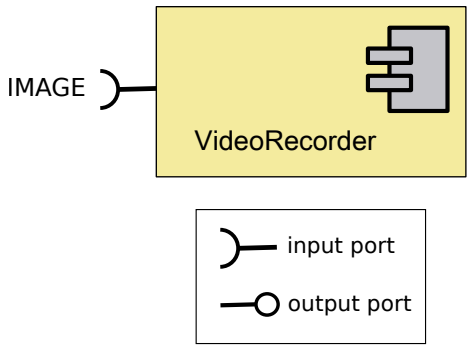

Figura 10.16: Componente VideoRecorder

A continuación se muestra el código CoolBOT a partir del cual es posible extraer las características y parámetros del componente VideoRecorder.

Código CoolBOT que representa al componente VideoRecorder

```
/*
 * File: video-recorder.coolbot-component
 * Description: description file for VideoRecorder component
 * Date: 14 May 2012
 * Generated by coolbot-bundle version 1.1.0
 */
component VideoRecorder
{
    header
    {
        author "Antonio Carlos Dominguez Brito <adominguez@iusiani.ulpgc.es>"
            ;
```

```
        description "component VideoRecorder records a video of RGB frames
             provided by component Camera";
        institution "IUSIANI - Universidad de Las Palmas de Gran Canaria";
        version "0.1"
    };

    constants
    {
      DEFAULT_FPS = 25; // frames per second
      private FILE_TIMESTAMP_N_DIGITS = 15;
    };

    // input ports
    input port IMAGE type poster port packet PacketRGBImage;

    entry state loop
    {
      transition on IMAGE;
    };
};
```

Este componente ha sido diseñado para recibir imágenes en formato RGB de otro componente y generar con todas las recibidas un archivo de vídeo de formato "AVI"[2], formato contenedor de audio y vídeo bastante extendido. La frecuencia del vídeo generado corresponde a la calculada en función de los frames por segundo (FPS) especificados al componente o de los FPS por defecto del código CoolBOT.

Internamente utiliza la biblioteca OpenCV[28] para crear el vídeo y almacenar las imágenes. Finalizada su ejecución genera un vídeo cuyo nombre presenta el formato "nombre-de-video.avi", siendo el nombre del vídeo el introducido por el usuario en su ejecución.

10.2.2. Integración camera-ip-axis-recorder

Para lograr generar archivos de vídeo a partir de imágenes de la cámara IP se crea una integración compuesta por los componentes CameraIPAxis y VideoRecorder. Adicionalmente se incluye la vista CameraView para visualizar las imágenes mientras se produce la grabación del vídeo. La integración toma el nombre de **camera-ip-axis-recorder** y se desarrolla en el coolbot-camera-ip-bundle.

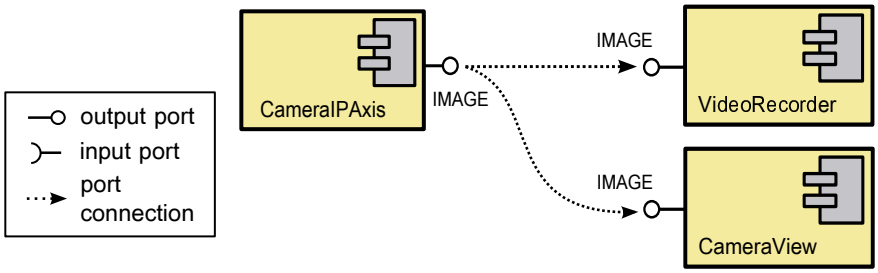

Figura 10.17: Integración camera-ip-axis-recorder.

Como se observa en la figura 10.17 tanto el componente VideoRecorder como la vista CameraView están conectados al puerto de salida de CameraIPAxis. Esto permite ver las imágenes de la cámara IP al mismo tiempo que son almacenadas.

A continuación se puede ver el código CoolBOT de la integración donde se especifican las instancias de los componentes y la vista, así como las conexiones existentes entre ellos.

Código CoolBOT que representa la integración camera-ip-axis-recorder

```
/*
 * File: camera-ip-axis-recorder.coolbot-integration
 * Description: description file for camera-ip-axis-recorder integration.
 * Date: 29 September 2012
 * Generated by coolbot-bundle version 1.1.0
 */

integration camera-ip-axis-recorder
{

  header
  {
    author "Bycor Sanchez Sanchez <bycor.sanchez@gmail.com>";
    description "Integration including a CameraIPAxis instance and a
         CameraRecorder instance";
    institution "IUSIANI - Universidad de Las Palmas de Gran Canaria";
    version "0.1"
  };

  machine addresses
  {
    // Machine addresses definition.
    local dis172ac: "dis172acplg.dis.ulpgc.es";
  };

  local instances
  {
    // Local instances definition.
    component theCamera:CameraIPAxis;
    component theRecorder:VideoRecorder;

    view theView:CameraView with description "Camera IP Axis View";
  };

  port connections
  {
    connect theCamera:IMAGE to theView:IMAGE;
    connect theCamera:IMAGE to theRecorder:IMAGE;
  };

};
```

Al ser ejecutada, la integración pide al usuario la dirección IP o dominio de red de la cámara IP Axis a la que desea conectarse, usuario y contraseña de autentificación y finalmente el nombre que se desea poner al archivo de vídeo.

En caso de error se mostrará en pantalla la posible causa del mismo. Bajo un correcto funcionamiento se genera una ventana donde se muestran las imágenes en directo que emite la cámara IP al mismo tiempo que son grabadas. El botón "close" situado en la parte inferior de la ventana finaliza la ejecución, generando en dicho momento el archivo de vídeo. En la figura 10.18 se puede ver un ejemplo de su funcionamiento.

El nombre del vídeo generado presenta el formato "nombre-de-video.avi", siendo el nombre del vídeo el introducido al inicio de la ejecución de la integración camera-ip-axis-recorder.

Figura 10.18: Ejemplo de funcionamiento camera-ip-axis-recorder

10.3. Sistema de vídeo distribuido

El tercer objetivo planteado en este TFG es el desarrollo de un sistema de gestión capaz de reproducir y ofrecer un acceso integrado a las secuencias de vídeo grabadas por el sistema, de manera que la red de cámaras IP integradas pueda tener tanto una topología, como una composición variable.

Las funcionalidades que este sistema de gestión de vídeo aporta son:

1. **Reproducción** concurrente de **múltiples vídeos** generados por la infraestructura desarrollada en las etapas anteriores.

2. **Control de la reproducción**. Posibilidad de manejar los vídeos con los controles habituales como iniciar la reproducción, pararla, reiniciarla, incluso situar el vídeo en un instante de tiempo concreto.

3. Representación de los vídeos en una **escala de tiempo** que respete la temporalidad real de los mismos. Esto permite que visualmente sea posible estimar su duración así como observar la diferencia de tiempo transcurrida entre el inicio/fin de un vídeo con cualquier otro de los grabados.

4. Posibilidad de reproducir un **vídeo en pantalla completa**. Esta capacidad permite la omisión de la interfaz creada para visualizar con mayor detalle un vídeo en concreto.

En etapas previas se ha diseñado un boceto con una interfaz que aporta las listadas funcionalidades. La figura 10.19 corresponde a dicho boceto.

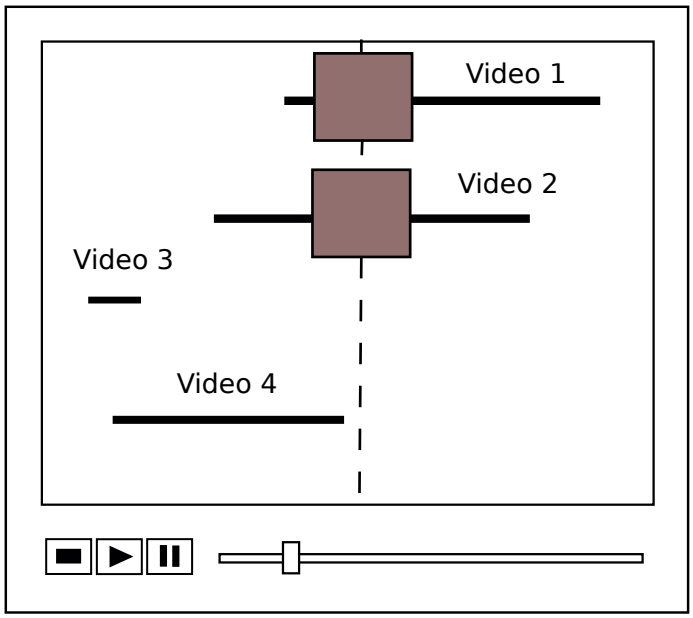

Figura 10.19: Diseño inicial.

El "bundle" **coolbot-camera-bundle**, espacio de trabajo con software CoolBOT relacionado con *cámaras web* o *webcams*, contiene una serie de componentes que ya permiten la reproducción de un vídeo con una interfaz propia. Estos componentes servirán como referencia para desarrollar el sistema de vídeo distribuido.

10.3.1. Referencia

Durante la creación del sistema distribuido de vídeo se toma como referencia el componente **VideoPlayer** y la vista **VideoPlayerView**. Ambos componentes software pertenecen al **coolbot-camera-bundle** y permiten la reproducción de un vídeo, para lo cual interactúan entre sí mostrando una interfaz y controlando las acciones del usuario.

10.3.1.1. Paquetes VideoPlayerPackets

Con el fin de transmitir información entre componentes CoolBOT encapsula los datos a enviar en unidades discretas llamadas "port packets". Éstos representan un formato adecuado para la transmisión y permiten la extracción de los datos una vez recibidos.

Para los tipos de datos más utilizados y comunes CoolBOT dispone de "port packets" predefinidos, algunos de ellos representados en la tabla 9.2. Sin embargo, para nuevos tipos de datos CoolBOT permite la creación de nuevos "port packets".

En este caso el paquete VideoPlayerPackets de **coolbot-camera-bundle** crea tres nuevos "port packets" para la transmisión de nuevos tipos de datos por parte del componente VideoPlayer y la vista VideoPlayerView.

Código CoolBOT que representa al paquete VideoPlayerPackets

```
/* File: video-player-packets.coolbot-packets
 * Description: description file for new packets definition.
 * Date: 15 May 2012
 * Generated by coolbot-bundle version 1.1.0
 */
packets VideoPlayerPackets
{
  header
  {
    author "Antonio Carlos Dominguez Brito <adominguez@iusiani.ulpgc.es>";
    description "VideoPlayer port packets";
    institution "IUSIANI - Universidad de Las Palmas de Gran Canaria - Spain"
        ;
    version "0.1"
  };

  port packet CommandPacket
  {
    data members
    {
      id: type "unsigned short"; // command identifier
      millis: type "long"; // video time in milliseconds
    };
  };

  port packet ConfigPacket
  {
    data members
    {
      fps: type "double"; // frames per second
      width: type "unsigned int"; // frame width (pixels)
      height: type "unsigned int"; // frame height (pixels)
      frames: type "unsigned int"; // number of frames in the video file
      totalTime: type "Time"; // video time lenght
      eovf: type "bool"; // end of video file reached?
      videoFile: type "string";
    };
  };

  port packet FramePacket
  {
    data members
    {
      image: type "RGBGrid";
      index: type "unsigned int"; // frame index
      relativePosition: type "double"; // relative position on the video
    };
  };
};
```

A partir de su código CoolBOT, adjunto anteriormente, se observa como los tres nuevos "port packets" son: CommandPacket, ConfigPacket y FramePacket. A continuación se explica la finalidad de cada uno, así como qué representan sus diferentes parámetros.

CommandPacket. Este "port packet" permite la transmisión de información relativa al tipo de comando seleccionado en la interfaz de usuario de la vista VideoPlayerView, sección 10.3.1.3. Consta de dos datos miembro:

- *id*, tipo "unsigned short". Identificador de comando.
- *millis*, tipo "long". Posición de la barra de tiempo, expresada en milisegundos. Es utilizado cuando el comando indicado modifica el instante actual de la reproducción (barra temporal de la interfaz gráfica).

ConfigPacket. "Port packet" que permite transmitir los datos de configuración propios de un vídeo. Está compuesto por los siguientes datos miembro:

- *videoFile*, tipo "string". Nombre del archivo de vídeo.
- *fps*, tipo "double". Frames por segundo (FPS), imágenes por segundo.
- *width*, tipo "unsigned int". Ancho de la imagen.
- *height*, tipo "unsigned int". Alto de la imagen.
- *frames*, tipo "unsigned int". Cantidad total de imágenes que dispone el vídeo.
- *totalTime*, tipo "Time". Duración del vídeo, "Time" es un tipo de datos de CoolBOT para expresar tiempo de manera precisa.
- *eovf*, tipo "bool". Indica si se ha alcanzado el final del vídeo.

FramePacket. Este "port packet" permite la transmisión de una imagen en formato RGB, así la posición que ocupa dentro del vídeo. Consta de tres datos miembro:

- *image*, tipo "RGBGrid". Imagen en formato RGB, el tipo de datos "RGBGrid" de CoolBOT permite almacenar y tratar este tipo de imágenes.
- *index*, tipo "unsigned int". Índice de la imagen dentro del vídeo.
- *relativePosition*, tipo "double". Posición relativa en el vídeo.

10.3.1.2. Componente VideoPlayer

El componente VideoPlayer se encuentra en el **coolbot-camera-bundle**. Dispone de un puerto de entrada llamado `COMMANDS` el cual recibe paquetes de información, "port packets", específicos para comandos.

Dispone también de dos puertos de salida, a través del puerto `FRAME` el componente envía imágenes y por el puerto `VIDEO_CONFIG` se envían las características propias del vídeo en reproducción. La figura 10.20 representa este componente.

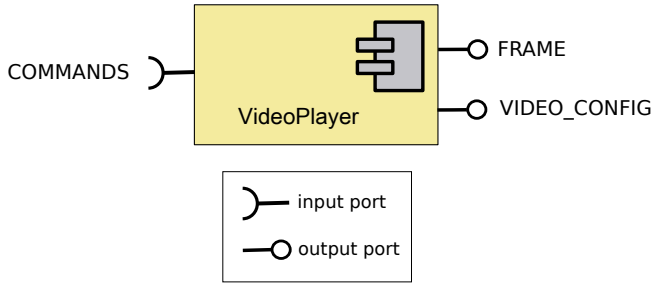

Figura 10.20: Componente VideoPlayer

La información referente a este componente se encuentra en su código CoolBOT adjunto a continuación.

Código CoolBOT que representa al componente VideoPlayer

```
/* File: video-player.coolbot-component
 * Description: description file for VideoPlayer component
 * Date: 15 May 2012
 * Generated by coolbot-bundle version 1.1.0
 */
component VideoPlayer
{
    header
    {
        author "Antonio Carlos Dominguez Brito <adominguez@iusiani.ulpgc.es>"
            ;
        description "component VideoPlayer plays a video of RGB frames
            recorded using VideoRecorder";
        institution "IUSIANI - Universidad de Las Palmas de Gran Canaria";
        version "0.1"
    };

    constants
    {
      private FIFO_LENGTH=5;
    };

    // input ports
    input port COMMANDS type fifo port packet VideoPlayerPackets::
        CommandPacket length FIFO_LENGTH;

    //output ports
    output port VIDEO_CONFIG type poster port packet VideoPlayerPackets::
        ConfigPacket;
    output port FRAME type poster port packet VideoPlayerPackets::FramePacket
        ;

    entry state loop
    {
      transition on COMMANDS,TIMER;
    };
};
```

A partir de un vídeo grabado VideoPlayer controla qué imágenes mostrar en cada momento por la interfaz gráfica. Debido a que la reproducción está sujeta a una serie de comandos (inicio, parar, reiniciar, etc) es necesario controlar su estado para determinar las imágenes a enviar. Para ello el componente se vale de un puerto de entrada llamado `COMMANDS` el cual recibe los comandos seleccionados por el usuario.

Adicionalmente el componente suministra a través del puerto de salida `VIDEO_CONFIG` parámetros de configuración propios del vídeo como son su nombre, duración, frecuencia, entre otros.

Los tipos de datos que recibe y envía este componente no poseen "port packets" predefinidos en CoolBOT, debido a lo cual previamente se han definido nuevos "port packets" específicos en el espacio de nombres C++ VideoPlayerPackets, sección 10.3.1.1.

10.3.1.3. Vista VideoPlayerView

La vista VideoPlayerView perteneciente al **coolbot-camera-bundle**. Ofrece una interfaz gráfica que permite al usuario controlar la reproducción de un vídeo grabado.

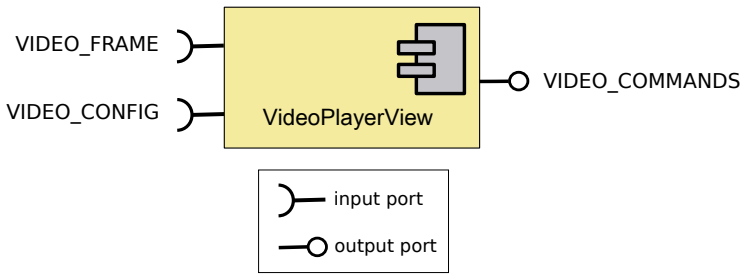

Figura 10.21: Vista VideoPlayerView

Esta vista posee un puerto de salida, `VIDEO_COMMANDS`, a través del cual comunica el comando o control seleccionado por el usuario en la interfaz gráfica.

Dispone además de dos puertos de entrada, `VIDEO_FRAME` el cual recibe la imagen a mostrar y `VIDEO_CONFIG` que obtiene los detalles propios del vídeo como su nombre, duración, frecuencia, etc.

A continuación se incluye el código CoolBOT correspondiente a la vista VideoPlayerView. Es posible observar los puertos de entrada y salida ya nombrados, además de ciertos parámetros que definen el aspecto de la interfaz gráfica de usuario.

Código CoolBOT que representa a la vista VideoPlayerView

```
/*
 * File: video-player-view.coolbot-view
 * Description: description file for VideoPlayerView view
 * Date: 16 May 2012
 * Generated by coolbot-bundle 1.1.0
 */
view VideoPlayerView
{
  header
  {
    author "Antonio Carlos Dominguez-Brito <adominguez@iusiani.ulpgc.es>";
    description "VideoPlayerView view";
    institution "IUSIANI - Universidad de Las Palmas de Gran Canaria - Spain"
        ;
    version "0.1"
  };

  constants
  {
    private DEFAULT_REFRESHING_PERIOD=40; // milliseconds
    private DEFAULT_SIZE_X=640; // pixels
    private DEFAULT_SIZE_Y=480; // pixels
    private DEFAULT_TEXT_MARGIN=2; // pixels
    private TILT_SCALE_INCREMENT=5; // degrees
  };

  // input ports
  input port VIDEO_CONFIG type poster port packet VideoPlayerPackets::
      ConfigPacket;
  input port VIDEO_FRAME type poster port packet VideoPlayerPackets::
      FramePacket;

  // output port
  output port VIDEO_COMMANDS type generic port packet VideoPlayerPackets::
      CommandPacket;
};
```

La interfaz gráfica, visible en la figura 10.22, está compuesta por la imagen del vídeo, información correspondiente al tiempo transcurrido, panel de controles, barra temporal y botón de finalización.

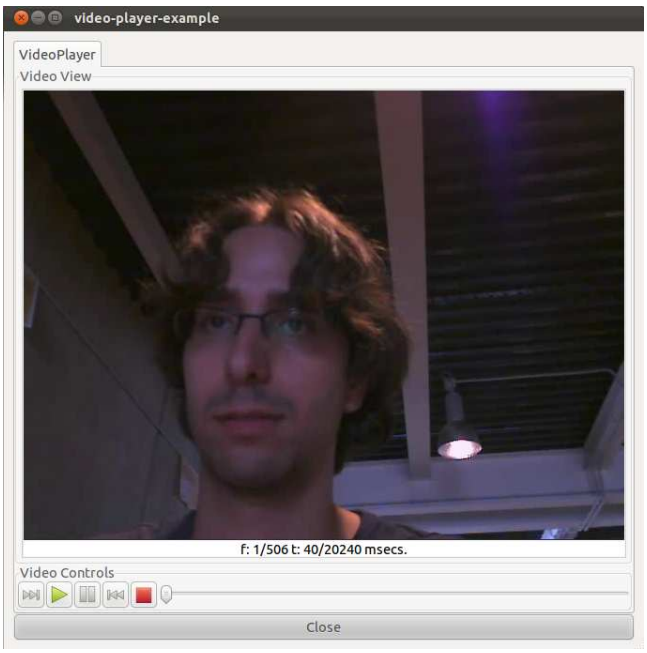

Figura 10.22: Interfaz VideoPlayerView

Para el envío y recepción de datos se utilizan los "port packets" creados específicamente para ello en el paquete VideoPlayerPackets. La información del paquete se encuentra en la sección 10.3.1.1.

10.3.1.4. Integración video-player-example

Para lograr la reproducción de un vídeo la integración **video-player-example** crea un sistema interconectado entre el componente VideoPlayer y la vista VideoPlayerView, haciendo ambos uso de los "port packets" aportados por el paquete VideoPlayerPackets.

La integración está representada en la figura 10.23. Con ella se consigue una interacción entre la interfaz gráfica y el control del vídeo de tal forma que, cuando un usuario seleccione un comando, éste sea transmitido al componente VideoPlayer y actúe en consecuencia enviando las imágenes y configuración requeridas a la vista.

El código CoolBOT de esta integración, video-player-example, muestra las instancias existentes del componente VideoPlayer y la vista VideoPlayerView, al igual que las conexiones existentes entre sus distintos puertos.

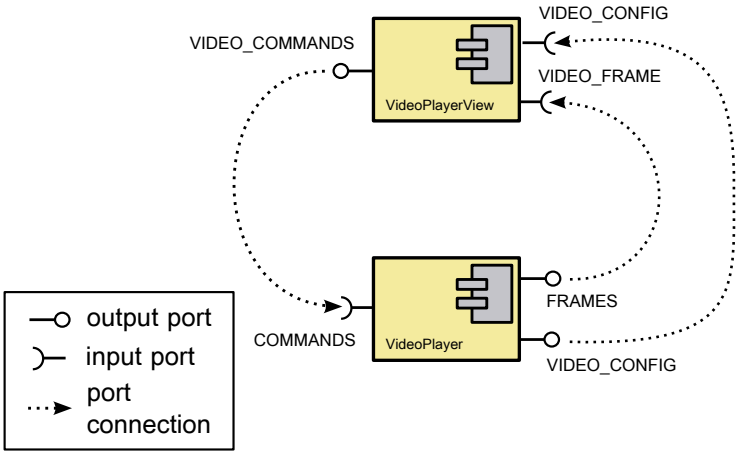

Figura 10.23: Integración video-player-example

Código CoolBOT que representa a la integración video-player-example

```
/*
 * File: video-player-example.coolbot-integration
 * Description: description file for video-player-example integration.
 * Date: 16 May 2012
 * Generated by coolbot-bundle version 1.1.0
 */
integration video-player-example
{
  header
  {
    author "Antonio Carlos Dominguez Brito <adominguez@iusiani.ulpgc.es>";
    description "This integration allows to player a video recorded using
         VideoRecorder";
    institution "IUSIANI - Universidad de Las Palmas de Gran Canaria";
    version "0.1"
  };

  machine addresses
  {
    local dis172ac: "dis172acplg.dis.ulpgc.es";
  };

  local instances
  {
    component thePlayer:VideoPlayer;
    view theView:VideoPlayerView with description "VideoPlayer";
  };

  port connections
  {
    connect thePlayer:FRAME to theView:VIDEO_FRAME;
    connect thePlayer:VIDEO_CONFIG to theView:VIDEO_CONFIG;
    connect theView:VIDEO_COMMANDS to thePlayer:COMMANDS;
  };
};
```

Un ejemplo de su funcionamiento es el siguiente. En la interfaz gráfica un usuario selecciona el control "play" ya que desea reproducir el vídeo, inmediatamente el comando es enviado al componente VideoPlayer. Tras la detección del comando el componente transmite a la vista las imágenes del vídeo con la frecuencia del mismo. Como resultado en la interfaz se mostrará cada una de las imágenes, dando lugar a la reproducción del vídeo.

La figura 10.24 muestra el funcionamiento de esta integración reproduciendo un vídeo grabado por el componente VideoRecorder, sección 10.2.1.

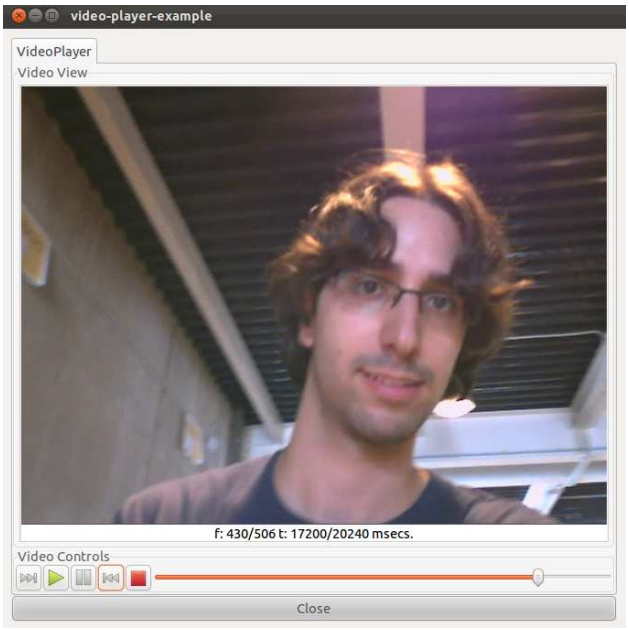

Figura 10.24: Interfaz video-player-example

10.3.2. Infraestructura software

El sistema de vídeo integrado a crear se caracteriza por ofrecer un **acceso integrado a secuencias de múltiples vídeos** grabados por un sistema de cámaras, principalmente cámaras IP. Además, **respeta la temporalidad de los vídeos** de tal manera que aquellas secuencias grabadas en el mismo instante de tiempo por diferentes cámaras puedan ser visibles conjuntamente.

Precisamente para garantizar la temporalidad de los vídeos se necesita conocer el momento exacto en el que fueron grabados. Por ello, antes de continuar con el desarrollo es necesario **modificar el componente VideoRecorder**, utilizado en el almacenamiento en vídeo, para guardar el instante de tiempo en el que se inicia la grabación.

Para mayor facilidad se ha decidido que el componente VideoRecorder almacene el momento exacto de la grabación en el propio nombre del vídeo generado. Como se especifica al final del apartado 10.2.1 el formato que adopta el nombre del vídeo contiene su nombre y la extensión del formato AVI[2], ejemplo "video.avi". Éste se verá modificado para añadir el inicio de la grabación como tiempo Unix[32], el cual representa un instante de tiempo en segundos transcurridos desde el 1 Enero del 1970, a las 00:00:00 UTC.

De este modo el **formato final del nombre** para los vídeos generados por el componente VideoRecorder se compone del nombre aportado por el usuario, carácter separador "-", 15 dígitos para la representación del instante de tiempo en tiempo Unix[32] y extensión del formato AVI[2]. Un ejemplo es el siguiente "video-001353068261365.avi".

Realizada esta modificación se procede a la creación de la infraestructura software para el sistema de vídeo integrado. Se utiliza como referencia los componentes software del **coolbot-camera-bundle** que permiten la reproducción de un vídeo, indicados en la sección 10.3.1.

La infraestructura mantiene una estructuración como la del software de referencia, compuesta por los siguientes elementos:

- **Paquetes** (MultiVideoPackets) que define nuevos "port packets" para el envío de múltiples imágenes y múltiples configuraciones.
- **Componente** (MultiVideoPlayer) capaz de controlar las acciones introducidas por el usuario en la interfaz gráfica y actuar en consecuencia. Entre sus cometidos está la extracción de los datos de configuración de cada vídeo, detectar qué vídeos poseen imágenes en un instante de tiempo concreto, envío de múltiples imágenes, entre otros.
- **Vista** (MultiVideoPlayerView) que ofrezca una interfaz sencilla al usuario para el manejo del sistema de vídeo integrado. En dicha interfaz se ofrece información visual acerca de parámetros globales, como la duración total de las grabaciones, así como información propia de cada vídeo como su duración, comparación temporal con respecto al resto de vídeos, etc.

Mediante la integración del componente y la vista, haciendo uso de los nuevos "port packets" en su comunicación, se obtendrá la infraestructura que garantice el tercer objetivo planteado en el presente TFG. En las siguientes secciones se desarrollan cada uno de los elementos.

10.3.2.1. Paquetes MultiVideoPlayerPackets

CoolBOT encapsula los datos en unidades discretas llamadas "port packets" que representan un formato adecuado para su transmisión entre componentes. Los tipos de datos más comunes y utilizados disponen de sus propios "port packets" , no obstante, los nuevos tipos de datos requieren la creación de nuevos paquetes de puerto.

Al crear un nuevo paquete, CoolBOT genera una clase de C++ por cada "port packet" contenido en el mismo, de tal forma que implementa por defecto características intrínsecas a los "port packets" como las operaciones de encapsulamiento y extracción de datos (marshalling y demarshalling).

En el espacio de trabajo **coolbot-camera-ip-bundle** los paquetes MultiVideoPlayerPackets definen dos nuevos "port packets" para la transmisión de datos en el sistema de vídeo integrado. A continuación se puede ver el código CoolBOT de MultiVideoPlayerPackets.

Código CoolBOT que representa al paquete MultiVideoPlayerPackets

```
/*
 * File: multi-video-packets.coolbot-packets
 * Description: description file for new packets definition.
 * Date: 22 October 2012
 * Generated by coolbot-bundle version 1.1.0
 */
packets MultiVideoPackets
{
  /* Packet's definition. */
  port packet ConfigPacket
  {
    data members
    {
      videos: type "VideoArray";
      totalTime: type "Time";    // total time lenght
      eov: type "bool";          // end of all videos reached?
    };
  };

  port packet FramesPacket
  {
    data members
    {
      frames: type "FrameArray";
      currentTime: type "Time";
    };
  };
};
```

Como se puede ver en el código CoolBOT asociado al MultiVideoPackets los nuevos "port packets" creados son:

ConfigPacket. Este nuevo "port packet" permite la transmisión de información de los vídeos que se reproducen, además de ciertos parámetros globales. Consta de los siguientes datos miembros:

- *videos*, tipo "VideoArray". Vector con los datos de configuración de cada uno de los vídeos.
- *totalTime*, tipo "Time". Duración total de los vídeos, tiempo transcurrido desde la grabación del primer vídeo hasta el fin de la grabación del último en el tiempo.
- *eov*, tipo "bool". Parámetro que indica si todos los vídeos han sido reproducidos.

El vector "VideoArray" es creado en la clase C++ ConfigPacket y para su implementación se han utilizado las plantillas proporcionados por CoolBOT para crear componentes de datos en paquetes. "VideoArray" está compuesto por dos elementos.

1. **Length**. Cantidad de elementos que almacena el vector, corresponde al número total de vídeos del sistema.
2. Estructura **VideoInfo**, utilizada para almacenar en cada posición de VideoArray los datos de configuración de un vídeo. La estructura VideoInfo contiene los siguientes datos:
   - *fps*, tipo "double". Frames por segundo, frecuencia del vídeo.

- *width*, tipo "unsigned int". Ancho de la imagen del vídeo.
- *height*, tipo "unsigned int". Alto de la imagen del vídeo.
- *frames*, tipo "unsigned int". Cantidad de imágenes que posee el vídeo.
- *initTime*, tipo "Time". Instante de tiempo del inicio del vídeo.
- *totalTime*, tipo "Time". Duración total del vídeo.
- *videoFile*, tipo "string". Nombre del archivo de vídeo.

FramesPacket. Este nuevo "port packet" permite transmitir de una sola vez múltiples imágenes en un único paquete. Adicionalmente indica el instante de tiempo al que pertenecen dichas imágenes. Los datos miembros que conforman el FramesPacket son:

- *frames*, tipo "FrameArray". Vector que almacena las diferentes imágenes a enviar.
- *currentTime*, tipo "Time". Parámetro que indica a qué instante de tiempo pertenecen las imágenes transportadas.

El vector "FrameArray" es creado en la clase C++ FramesPacket y para su implementación se han utilizado las plantillas proporcionados por CoolBOT para crear componentes de datos en paquetes. "FrameArray" se compone de dos elementos.

1. **Length**. Cantidad de elementos que almacena el vector, corresponde al número de vídeos que envían imágenes.
2. Estructura **VideoFrame**, utilizada para almacenar en cada posición del vector FrameArray la imagen a enviar. La estructura VideoFrame contiene los siguientes elementos:
   - *image*, tipo "RGBGrid". Imagen en formato RGB. El tipo "RGBGrid" es propio de CoolBOT para este tipo de imágenes.
   - *index*, tipo "unsigned int". Posición de la imagen dentro del vídeo.
   - *video_src*, tipo "int". Vídeo al que pertenece la imagen enviada.

10.3.2.2. Componente MultiVideoPlayer

El componente **MultiVideoPlayer** aporta la funcionalidad al sistema de vídeo integrado y constituye el núcleo de esta infraestructura software. Entre sus cometidos destaca el acceso a los vídeos grabados en etapas anteriores, control temporal de la reproducción, envío de múltiples imágenes de manera simultánea, etc. Al igual que el resto de componentes relacionados con cámaras IP se encuentra en el espacio de trabajo **coolbot-camera-ip-bundle**.

Este componente interactúa con una interfaz gráfica para dar al usuario el control sobre los vídeos grabados. Por dicho motivo dispone de una serie de puertos que permiten la comunicación entre ambos componentes software como se puede observar en la figura 10.25. En concreto posee un puerto de entrada, llamado

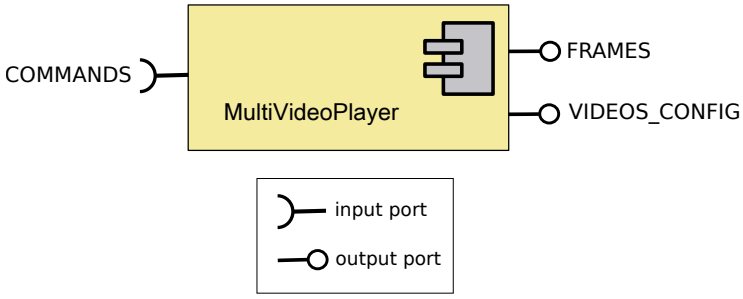

Figura 10.25: Componente MultiVideoPlayer

COMMANDS, a través del cual recibe las acciones o comandos introducidos en la interfaz gráfica, como por ejemplo iniciar la reproducción, pararla, situarse en un instante de tiempo determinado, entre otros.

Como puertos de salida dispone del puerto VIDEOS_CONFIG, que suministra datos de configuración de cada vídeo, y del puerto FRAMES, capaz de enviar de forma simultánea las imágenes de múltiples vídeos en un instante de tiempo concreto.

A continuación se muestra el código CoolBOT a partir del cual se genera el componente.

Código CoolBOT que representa al componente MultiVideoPlayer.

```
/*
 * File: multi-video-player.coolbot-component
 * Description: description file for MultiVideoPlayer component
 * Date: 11 October 2012
 * Generated by coolbot-bundle version 1.1.0
 */
component MultiVideoPlayer
{
  header
  {
    author "Bycor Sanchez Sanchez <bycor.sanchez@gmail.com>";
    description "MultiVideoPlayer CoolBOT component - this component sends
         multiple frames from multiple videos on real time";
    institution "IUSIANI - Universidad de Las Palmas de Gran Canaria";
    version "0.1"
  };

  constants
  {
    private FIFO_LENGTH=5;
  };

  // input ports
  input port COMMANDS type fifo port packet VideoPlayerPackets::CommandPacket
       length FIFO_LENGTH;

  //output ports
  output port VIDEOS_CONFIG type poster port packet MultiVideoPackets::
      ConfigPacket;
  output port FRAMES type poster port packet MultiVideoPackets::FramesPacket;

  entry state loop
  {
    transition on COMMANDS,TIMER;
  };
};
```

A partir del código CoolBOT se pueden extraer ciertos aspectos del diseño del componente MultiVideoPlayer como los siguientes.

Los comandos recibidos por el puerto `COMMANDS` hacen uso del "port packet" *ConfigPacket* para la transmisión de los datos. Este "port packet" está definido en el paquete VideoPlayerPackets del software de referencia, sección 10.3.1.1.

El puerto `COMMANDS` es un puerto de entrada FIFO (First In First Out) que permite encolar por orden de llegada los comandos recibidos de la interfaz gráfica.

El puerto de salida `VIDEOS_CONFIG` utiliza para el envío de datos de configuración el "port packet" *ConfigPacket* definido en el paquete MultiVideoPlayerPackets de la sección 10.3.2.1.

El puerto de salida `FRAMES` hace uso del "port packet" *ConfigPacket* para el envío de múltiples imágenes. Dicho "port packet" está definido en el paquete MultiVideoPlayerPackets de la sección 10.3.2.1.

Su autómata posee un estado principal llamado "Loop" el cual dispone de dos transiciones. Una de las transiciones ocurre cuando salta el timer y la otra cuando se recibe un nuevo comando por el puerto `COMMANDS`.

Conocido ya en algo más de profundidad se determina el **funcionamiento que desempeña** el componente **MultiVideoPlayer**.

1. En primer lugar tratar los archivos de vídeo suministrados para **almacenar los datos relevantes**, así como determinar la duración total de la reproducción y conocer el mayor periodo de todos ellos.

2. **Ajustar el timer** interno al periodo de tiempo menor obtenido de entre todos los vídeos. De esta forma la velocidad de reproducción se adapta a la del vídeo más rápido (con mayor tasa de imágenes por segundo, FPS).

3. Enviar de forma inicial los **parámetros de configuración** de todos los vídeos, además de las **imágenes correspondientes al instante inicial** en que fue grabado el primero de ellos. Esto permitirá que la interfaz gráfica se configure y muestre las primeras imágenes de ellos.

4. Finalmente, durante la etapa de ejecución, **enviar las imágenes** que correspondan al instante de tiempo de la reproducción y a los comandos enviados por el usuario a través de la interfaz gráfica. En caso de finalizar totalmente la reproducción se envía los parámetros de configuración para notificarlo.

Los pasos de este esquema de funcionamiento se han de adaptar al autómata por defecto que CoolBOT utiliza para el modelado de los componentes. En la figura 10.26 se observa como el autómata consta de múltiples estados y transiciones que corresponden a eventos disponibles por defecto.

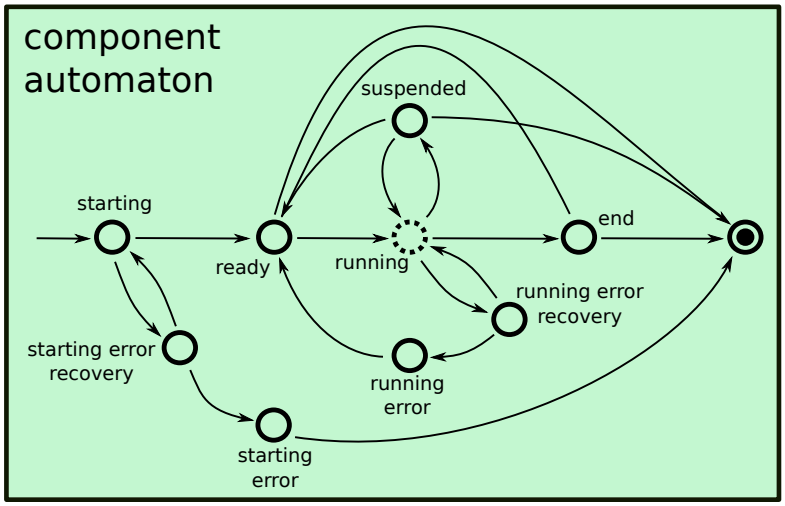

Figura 10.26: Autómata por defecto en componentes CoolBOT

Cabe destacar el metaestado **running**, el cual será sustituido por un **autómata diseñado por el usuario** que aporta la funcionalidad concreta al componente MultiVideoPlayer.

A continuación se plantean las acciones a realizar en los diferentes estados del autómata para adaptar la funcionalidad.

- **Almacenar los datos de los vídeos** suministrados por el usuario en el estado "starting". Será necesaria la creación de una función capaz de abrir cada vídeo, haciendo uso del tipo VideoCapture de OpenCV[28], y extraer la información relevante.
- **Inicializar los parámetros** restantes en el estado "starting".
- **Enviar los datos de configuración y las imágenes iniciales** en el estado "starting". Esto permitirá que la vista se configure y muestre las primeras imágenes.
- **Ejecución de un autómata propio** en el metaestado "running" que tenga en cuenta los siguientes aspectos.

  1. Reciba los comandos seleccionados en la interfaz de usuario y actúe en consecuencia. Por lo general la mayoría de los comandos envían por el puerto de salida `FRAMES` las imágenes disponibles en el instante de tiempo indicado.
  2. Sea capaz de enviar las imágenes disponibles de los vídeos continuamente con un periodo determinado en caso de que el comando recibido sea el de reproducción.
  3. En caso de terminar la reproducción de todos los vídeos se envíen los datos de configuración por el puerto de salida `VIDEOS_CONFIG` para comunicarlo a la vista.

El **diseño del autómata** que controla los comandos recibidos y envía las imágenes disponibles de cada vídeo en el instante de tiempo concreto de la reproducción dispone de un estado. Dicho estado se llama "Loop" y posee dos transiciones como se puede ver en la figura 10.27.

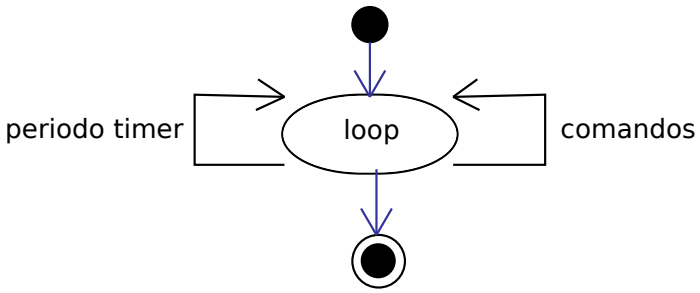

Figura 10.27: Autómata MultiVideoPlayer

La primera transición está asociada a un timer interno, por defecto desactivado, que permite el envío continuo de imágenes con una frecuencia calculada en función de a cuántas imágenes por segundo (FPS) se quieran reproducir los vídeos. La segunda transición, llamada "commands", se ejecuta cada vez que un nuevo comando es recibido y ha de actuar en consecuencia al mismo.

A pesar de pertenecer al mismo estado **cada transición se asocia con una función diferente**. A continuación se detalla en mayor medida el funcionamiento de las funciones asociadas.

Función loopTimer, asociada a la transición del timer. Esta función está diseñada para reproducir los vídeos cuando el usuario lo comunica por la interfaz gráfica.

Su funcionamiento consiste en **detectar qué vídeos poseen imágenes** para el siguiente instante de tiempo de la reproducción y **enviar** por el puerto de salida `FRAMES` **dichas imágenes**. Para la transmisión de múltiples imágenes se utiliza el "port packet" FramesPacket, definido en el paquete MultiVideoPackets de la sección 10.3.2.1. Una vez alcanzado el final de la reproducción, cuando todos los vídeos han sido reproducidos, se desactiva el timer.

Al estar asociada a un timer, esta función se ejecuta periódicamente cuando el timer se encuentra activo. La frecuencia con la que se ejecuta es configurada en función de las imágenes por segundo (FPS) que posea el vídeo más rápido. De esta forma se garantiza que todos los vídeos mantienen esa velocidad de reproducción.

La figura 10.28 representa el funcionamiento de la función loopTimer cada vez que es ejecutada.

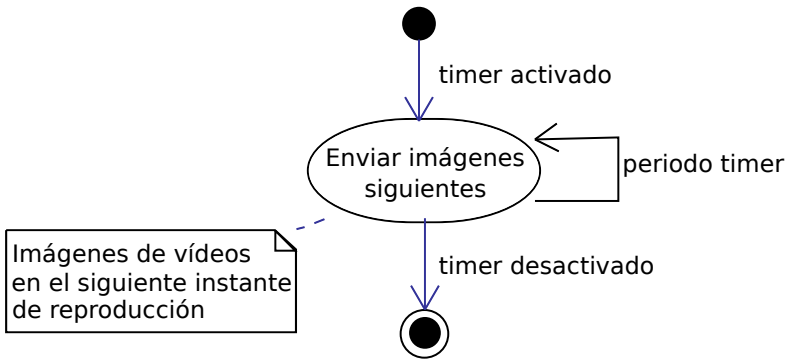

Figura 10.28: Funcionamiento loopTimer

Función loopCommands, asociada a la transición "commands". Esta función es ejecutada cada vez que se recibe un nuevo comando procedente de la vista. Estos comandos son recibidos en el puerto de entrada `COMMANDS` en "port packets" del tipo CommandPacket especificado en el paquete VideoPlayerPackets, sección 10.3.1.1.

El funcionamiento de "loopCommands" difiere según el tipo de comando recibido. Por ello, antes de continuar es necesario indicar los diferentes **tipos de comandos** existentes en la interfaz como son: reproducir, pausar, reiniciar, paso adelante, paso atrás e ir. A continuación se explica el funcionamiento de cada uno.

- *Reproducir* (play). Comando para la reproducción de los vídeos. En este caso la función "loopCommands" **activa el timer** con el periodo correspondiente al vídeo más rápido. Una vez activado, el timer enviará las imágenes disponibles cada vez que itere, en función del periodo indicado, utilizando su función asociada "loopTimer".

- *Pausar* (pause). Comando para pausar la reproducción en curso. Cuando se desea pausar la reproducción se **desactiva el timer**, lo que provoca el fin del envío de imágenes.

- *Reiniciar* (restart). Comando para reiniciar la reproducción al inicio. Para el reinicio es necesario **colocar todos los vídeos en su posición inicial** y **enviar las imágenes** de aquellos vídeos que dispongan de imágenes al inicio del tiempo global. Adicionalmente, son enviados los datos de configuración si antes del reinicio se había alcanzado el fin de la reproducción.

- *Paso adelante* (step forward). Comando utilizado para obtener las siguientes imágenes. En este caso solo es necesario **enviar las imágenes** de aquellos vídeos que disponen de una imagen en el siguiente instante de tiempo. Al alcanzar el final de la reproducción se envían los datos de configuración para comunicarlo.

- *Paso atrás* (step backward). Comando utilizado para obtener las imágenes anteriores. Para este comando se **envían las imágenes** de los vídeos que dispongan de una imagen en el anterior instante de tiempo a la reproducción.

- *Ir* (go). Comando utilizado para obtener imágenes de vídeos en un punto concreto de la reproducción. La función "loopCommands" **posiciona los vídeos** en el instante de tiempo indicado y **envía las imágenes** disponibles.

El funcionamiento de esta función se ve reflejado en la figura 10.29 adjunta.

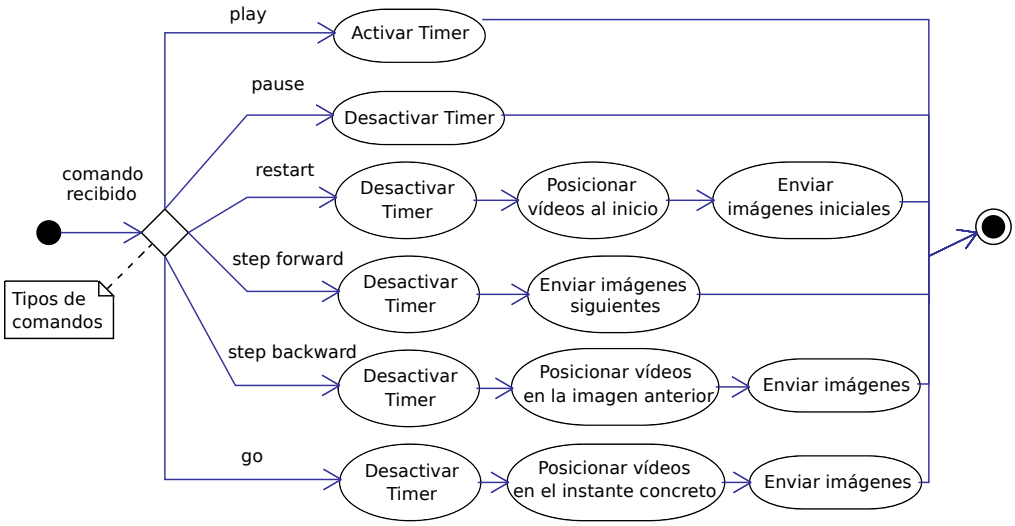

Figura 10.29: Funcionamiento loopCommands

A la hora de implementar el diseño planteado para el componente se hace necesario el desarrollo de una serie de elementos para el correcto funcionamiento.

1. **Tipos, estructuras y variables** para el almacenamiento de los datos más relevantes de cada vídeo, así como ciertos parámetros que afectan al componente.
2. **Función que inicialice** las estructuras y variables definidas que mantienen los datos relevantes de todos los vídeos.
3. **Función** que realice el **envío de datos de configuración**.
4. **Funciones para el envío de imágenes** disponibles en el instante de la reproducción.

Tipos, estructuras y variables de clase

Todo componente CoolBOT es implementado mediante una clase de C++[37]. Esta característica permite la definición numerosos elementos en la clase MultiVideoPlayer que permitirán almacenar datos de cada vídeo y parámetros que intervienen en el funcionamiento del componente. A continuación son detallados.

Tipo **lIInterval**. Este nuevo tipo es definido a partir de la clase "Interval" de CoolBOT, la cual ofrece una representación de un intervalo entre dos elementos

numéricos. Dicha clase aporta una serie de funciones relacionadas con intervalos como, por ejemplo, determinar sin un elemento se encuentra en él. El tipo "llInterval" utiliza elementos "long long" ya que representará un intervalo de tiempo con precisión.

Struct **VideoInfo**. Estructura definida en la clase MultiVideoPlayer que contiene información relativa a un vídeo. Entre sus elementos se encuentran:

- *name*, tipo "string". Indica el nombre del archivo de vídeo.
- *period*, tipo "long". Representa el periodo del vídeo actual, es decir, cada cuanto tiempo emite una nueva imagen. Se encuentra expresado en milisegundos.
- *interval*, tipo "llInterval". Este parámetro almacena el intervalo de tiempo en el que transcurre el vídeo. El tipo de datos es el definido anteriormente.
- *pVideoCapture*, tipo "VideoCapture". Parámetro que concede el acceso al vídeo. El tipo de datos VideoCapture corresponde a una clase de la biblioteca OpenCV[28] para la captura de vídeo, en este caso desde un archivo de vídeo. Dispone de multitud de funciones que permiten la extracción de una imagen, lectura de datos de configuración, entre otras funcionalidades.

Variables miembro de la clase.

- *videos*, vector de VideoInfo. Este vector contiene la información correspondiente a todos los vídeos suministrados por el usuario. Por defecto, la longitud del vector es igual al número máximo de vídeos permitidos.
- *period*, tipo "Time". Periodo del vídeo más rápido. El timer interno se ajustará a este periodo para garantizar que la reproducción de los vídeos se ajusta a la velocidad del más rápido.
- *numVideos*, tipo "int". Número de vídeos introducidos.
- *eov*, tipo "bool". Esta variable, abreviación de "end of videos", indica si ha finalizado la reproducción de todos los vídeos.
- *currentTime*, tipo "long long". Corresponde al instante actual en el que se encuentra la reproducción. El tipo "long long" permite representar el tiempo con precisión.
- *wholeInterval*, tipo "llInterval". Intervalo de tiempo desde la grabación del primer vídeo hasta el fin de la grabación del último en el tiempo. Representa el intervalo global de tiempo de todos los vídeos.
- *framesToSend*, lista de enteros. Esta lista almacena elementos tipo "int" y facilitará la detección y envío de las imágenes a enviar en cada momento.

Todas los elementos mencionados son privados a la clase MultiVideoPlayer y, por tanto, sólo son accesibles para funciones miembro de la misma clase.

Función inicializadora

Esta función es llamada desde el estado "starting" del autómata por defecto, figura 10.26, para inicializar las variables definidas en el apartado anterior.

Para la apertura y obtención de las características de cada vídeo se ha utilizado la clase VideoCapture de OpenCV[28]. Ésta permite una lectura de los vídeos haciendo uso de una serie de funciones, lo cual facilita la tarea de inicialización.

La función inicializadora toma el nombre de "**openMultiVideoFiles**". Se encarga de almacenar los datos de todos los vídeos en el vector `videos`, para lo cual rellena la estructura VideoInfo con los parámetros de cada uno. La información es obtenida gracias a la función `get`, perteneciente a la clase VideoCapture, que permite conocer las propiedades del vídeo.

De forma excepcional, el intervalo de tiempo de cada vídeo, parámetro `interval` de la estructura VideoInfo, se inicializa de forma diferente. El inicio de dicho intervalo corresponde al comienzo la grabación, información que se extrae del nombre del propio archivo, mientras que el final se deduce a partir del inicio y la duración del vídeo.

A medida que los datos son inicializados se realiza el cálculo para determinar el intervalo de tiempo global, que representará el tiempo total de reproducción, variable `wholeInterval`. Además, también se determina el periodo correspondiente al vídeo con mayor número de frames por segundo (FPS), vídeo más rápido, que determinará la velocidad de la reproducción.

Envío de datos de configuración

Para el envío de datos de configuración se ha creado una función llamada "**readAndSendConfiguration**". Esta función se encarga de almacenar en un "port packet" tipo ConfigPacket la información de todos los vídeos. Este "port packet" fue creado en el paquete MultiVideoPackets de la sección 10.3.2.1.

Los parámetros de configuración de cada vídeo son extraídos de la clase VideoCapture, utilizando para ello la función `get`. Los parámetros globales que conforman el ConfigPacket se obtienen de variables miembro de la clase MultiVideoPlayer.

La mayoría de los datos que conforman el ConfigPacket son constantes, no varían a lo largo de la ejecución, excepto el que advierte de la finalización total de la reproducción. Por esta razón el envío de datos de configuración se realiza cuando este parámetro varía, además de al inicio de la ejecución para configurar la interfaz de usuario (en el estado "starting").

Envío de imágenes

La implementación del envío de múltiples imágenes en el sistema de vídeo distribuido requiere tener en cuenta dos importantes aspectos como son:

1. Detección de vídeos con imágenes. La reproducción en el sistema de vídeo distribuido es compartida entre todos los vídeos, es decir, todos aquellos que dispongan de imágenes en un instante de tiempo serán reproducidos a la vez en la interfaz gráfica.

   Este diseño obliga a detectar qué vídeos poseen imágenes en el momento en el que se desean enviar, representado por la variable miembro `currentTime`. Para la detección de imágenes se han creado dos funciones.

   - Función "**fillList**". Esta función recorre todos los vídeos comprobando si el instante de tiempo actual, `currentTime`, se encuentra dentro del intervalo de reproducción de cada vídeo. En caso de ser así se determina que el vídeo posee imágenes a enviar.

     El identificador de cada vídeo detectado se incluye en la lista `framesToSend` para proceder más adelante al envío.

   - Función "**fillListAndSetVideo**". Al igual que la anterior, esta función recorre los vídeos comprobando si el instante de tiempo actual, `currentTime`, se encuentra dentro del intervalo de reproducción de cada vídeo. En caso de ser así incluye el identificador de cada vídeo detectado en la lista `framesToSend` para realizar posteriormente el envío.

     La diferencia con la función "fillList" reside en que el tiempo actual, `currentTime`, no corresponde al siguiente instante de la reproducción. Esto propicia que sea necesario posicionar el vídeo en el instante de tiempo requerido haciendo uso de la función `set` de VideoCapture.

     Esta función es utilizada en el envío de imágenes de los comandos de "reproducir" (play), "paso atrás" (step backward) e "ir" (go).

2. Enviar imágenes. Se ha implementado la función "**sendFrames**" para enviar las imágenes. Tras haber detectado los vídeos con imágenes a enviar, éstos se encuentran representados en la lista `framesToSend`.

   La función recorre dicha lista y rellena un "port packet" FramesPacket con las imágenes de los vídeos indicadas en ella. Las imágenes se obtienen mediante la función `read` de VideoCapture. Además, se rellena también el parámetro `currentTime` del FramesPacket para identificar el instante de tiempo al que pertenecen.

   Finalmente el "port packet" es enviado por el puerto `FRAMES` y se actualiza el tiempo actual de la reproducción, variable miembro de la clase `currentTime`. El "port packet" FramesPacket pertenece al paquete MultiVideoPackets de la sección 10.3.2.1.

10.3.2.3. Vista MultiVideoPlayerView

El sistema de vídeo distribuido requiere de una interfaz gráfica que permita al usuario su manejo. La vista **MultiVideoPlayerView** implementa esta interfaz tomando como modelo la vista del software de referencia VideoPlayerView. Al igual que el resto de componentes CoolBOT relacionados con cámaras IP la vista MultiVideoPlayerView está disponible en el espacio de trabajo **coolbot-camera-ip-bundle**.

A continuación se puede observar la vista MultiVideoPlayerView en la figura 10.30. Adicionalmente se adjunta su código CoolBOT.

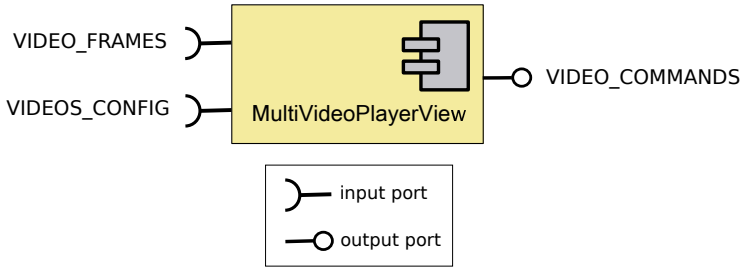

Figura 10.30: Vista MultiVideoPlayerView

Código CoolBOT que representa a la vista MultiVideoPlayerView.

```
/*
 * File: multi-video-player-view.coolbot-view
 * Description: description file for MultiVideoPlayerView view
 * Date: 08 October 2012
 * Generated by coolbot-bundle 1.1.0
 */
view MultiVideoPlayerView
{
  header
  {
    author "Bycor Sanchez Sanchez <bycor.sanchez@gmail.com>";
    description "MultiVideoPlayerView CoolBOT view";
    institution "IUSIANI - Universidad de Las Palmas de Gran Canaria";
    version "0.1"
  };

  constants
  {
    private DEFAULT_REFRESHING_PERIOD=100; // milliseconds
    private DEFAULT_SIZE_X=640; // pixels
    private DEFAULT_SIZE_Y=480; // pixels
    private DEFAULT_RESIZE_FACTOR=5; // factor to resize image
    private INTERMEDIATE_SPACE=55; // space between videos (pixels)
    private MAX_NUM_CHARS=13; // number of characters to display the video file name
    private DEFAULT_LINE_WIDTH=5;
    private DEFAULT_TEXT_MARGIN=5; // pixels
    private DEFAULT_GRIDS=6;
  };

  input port VIDEOS_CONFIG type poster port packet MultiVideoPackets::
      ConfigPacket;
  input port VIDEO_FRAMES type poster port packet MultiVideoPackets::
      FramesPacket;

  output port VIDEO_COMMANDS type generic port packet VideoPlayerPackets::
      CommandPacket;
};
```

A partir del código CoolBOT es posible extraer ciertos aspectos del diseño de la vista.

La vista MultiVideoPlayerView posee un gran número de constantes privadas que definen principalmente aspectos visuales de la interfaz como, por ejemplo, el tamaño de la pantalla, tamaño de las imágenes, ancho de las líneas, entre otros.

Excepcionalmente una constante, llamada `DEFAULT_REFRESHING_PERIOD`, define el periodo por defecto del refresco de la pantalla, es decir, cada cuanto tiempo se actualiza la información visual.

Esta vista posee dos puertos de entrada llamados `VIDEOS_CONFIG` y `VIDEO_FRAMES`. El primero de ellos, `VIDEOS_CONFIG`, recibe los datos de configuración de los vídeos en "port packets" del tipo ConfigPacket. El segundo , `VIDEO_FRAMES`, recibe las imágenes de los vídeos en "port packets" del tipo FramesPacket. Ambos "port packets" pertenecen a MultiVideoPackets, sección 10.3.2.1.

Finalmente se dispone de un puerto de salida llamado `VIDEO_COMMANDS` el cual informa del comando utilizado por el usuario en la interfaz gráfica. Estos comandos son enviados haciendo uso del "port packet" tipo CommandPacket perteneciente al paquete VideoPlayerPackets.

Las vistas CoolBOT no están diseñadas para realizar ningún tipo de funcionalidad, más allá de ofrecer una interfaz gráfica con la que interactuar con el usuario. Esta propiedad **simplifica el modelado de las vistas** pues sólo requiere **implementar la interfaz gráfica**. Para este cometido CoolBOT utiliza la biblioteca gráfica GTK+[13].

Tras la compilación del código CoolBOT de MultiVideoPlayerView, CoolBOT crea una clase correspondiente a la vista. En ella el compilador de CoolBOT genera una serie de estructuras básicas de GTK.

Dentro de las estructuras básicas el desarrollador generará sus propias estructuras GTK para crear la interfaz gráfica deseada. Algunas estructuras son capaces de generar eventos, por ejemplo, al pulsar un botón. Los eventos se asocian a funciones para definir qué acción llevar a cabo.

La vista MultiVideoPlayerView está compuesta por las siguientes estructuras GTK.

- *GtkScrolledWindow*. Ventana exterior de la interfaz que aportan barras de desplazamiento. La barras de desplazamiento sólo aparecen en caso de que la información visual no quepa en la ventana definida.

- *GtkLayout*. Ventana interior de la interfaz que ofrece la superficie sobre la cual mostrar la información visual de cada vídeo, así como sus imágenes. Posee los siguientes eventos asociados:

- Expose. Evento que controla la exposición de la información gráfica en la pantalla. Está asociado a la función "**imageExpose**" la cual obtiene los datos de los vídeos, a través de los datos de configuración e imágenes enviadas por el componente, y los imprime por la pantalla para ofrecerlos de forma visual. Con el fin de mantener actualizada la información el evento "expose" es provocado cada cierto tiempo con ayuda del timer por defecto de CoolBOT. El periodo con el que se actualiza se encuentra definido en la constante `DEFAULT_REFRESHING_PERIOD` del código CoolBOT.

- Size allocate. Evento que controla la modificación del tamaño de la ventana, es decir, se genera en caso de ampliar o disminuir la ventana. Está asociado a la función "**layoutSizeAllocate**", la cual reescala la ventana al nuevo tamaño.

- Button press. Evento que controla el click de ratón. En este caso el evento "button press" está asociado a la función "**doubleClick**". Esta función comprueba si se ha hecho doble click sobre el GtkLayout y si además la coordenada de donde se ha pulsado corresponde a la posición de una imagen. En caso de cumplirse comunica al evento "expose" que reproduzca en modo pantalla completa el vídeo seleccionado o que deje de hacerlo si ya lo hacía.

- *GtkButton*. Botones del panel inferior que ofrece el control de la reproducción. Cada botón representa una estructura GtkButton independiente. Dispone de un evento asociado:

  - Pressed. Evento que controla la pulsación en un botón. Cada botón de la interfaz está asociado a una función diferente que envía por el puerto `VIDEO_COMMANDS` un "port packet" del tipo CommandPacket con el comando correspondiente a ese botón. Por ejemplo, al pulsar el botón de reinicio su función asociada comunicará que el comando de reinicio fue seleccionado.

- *GtkHScale*. Barra horizontal inferior que controla el instante actual de la reproducción. Dispone de un evento asociado:

  - Change value. Evento que controla si la posición del cursor de la barra horizontal es modificado por el usuario. Al igual que el evento "pressed", está asociado a una función que envía por el puerto `VIDEO_COMMANDS` un "port packet" del tipo CommandPacket indicando que el comando seleccionado ha sido "ir". Adicionalmente indica la nueva posición que ocupa el cursor.

En la figura 10.31 se indica la posición que ocupan las diferentes estructuras GTK utilizadas en la interfaz del sistema de vídeo distribuido.

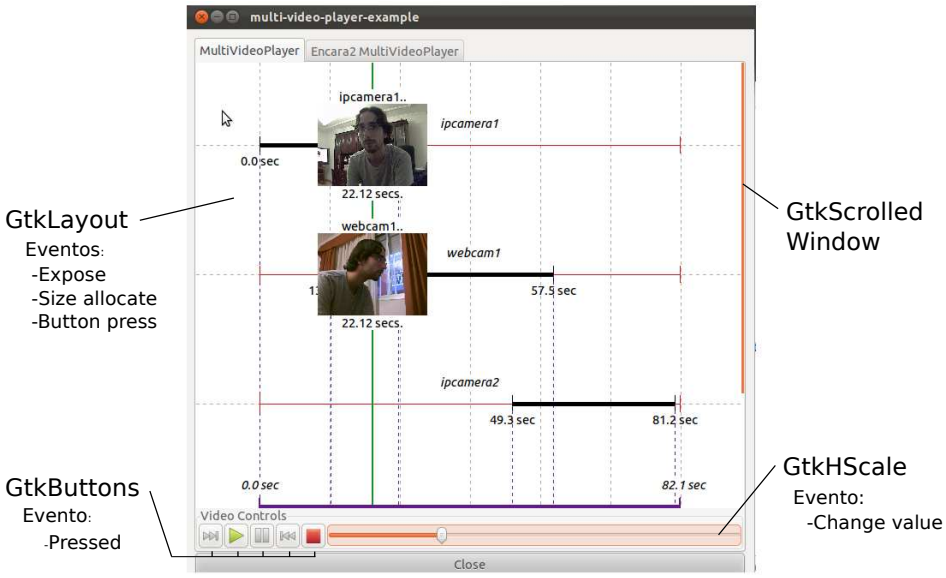

Figura 10.31: Posición estructuras GTK

La representación de los datos en la interfaz mantiene una **estructura visual** que facilita su correcta lectura y, al mismo tiempo, resulta intuitiva al usuario. En la creación de esta estructura se siguen una serie de pautas.

1. **Cada vídeo** es representado por su **nombre y una línea temporal**. En la línea de tiempo se destaca el tramo que corresponde al transcurso de dicho vídeo, es decir, desde el instante en que se inició la grabación hasta que finalizó. En los extremos de dicho tramo se incluye el instante exacto de inicio y fin.

2. Los tramos no destacados de la línea temporal, en color rojo, no pertenecen al vídeo y, por lo tanto, no dispone de imágenes.

3. De forma global se incluye en la parte inferior de la ventana una línea temporal adicional, de diferente color, que representa la **duración total de la reproducción**.

4. El instante actual de la reproducción se representa con una **línea vertical** de **color verde**.

5. Durante la reproducción de un vídeo las **imágenes se muestran escaladas**, no poseen su tamaño original, para respetar la información restante.

6. En caso de reproducir un vídeo en pantalla completa la **imagen** aparece en su **tamaño original**. Al pie de la imagen se ofrecen datos particulares al vídeo seleccionado.

La figura 10.32 ilustra la estructura visual de la interfaz indicando las propiedades explicadas.

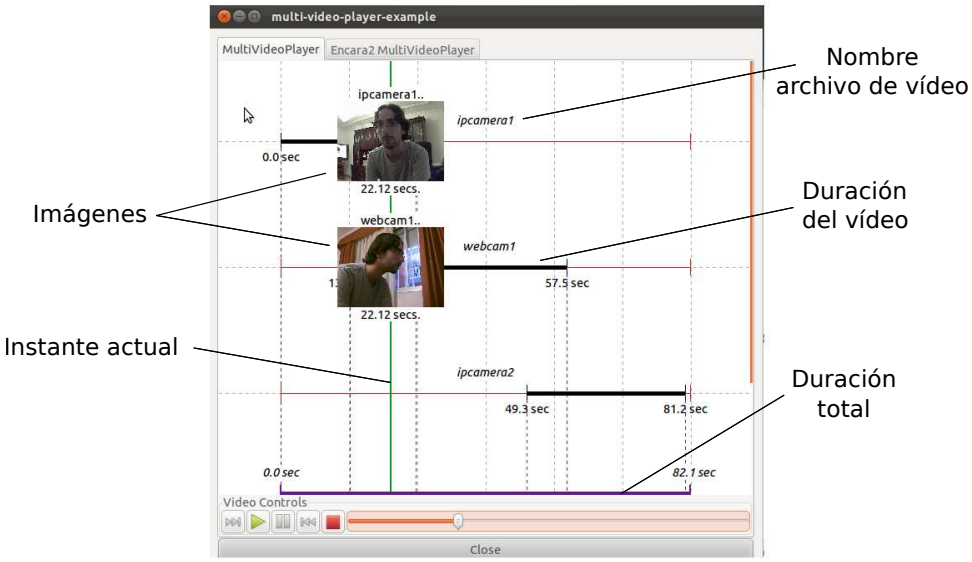

Figura 10.32: Interfaz gráfica de usuario

10.3.2.4. Integración multi-video-player-example

El desarrollo del sistema de vídeo distribuido finaliza con la creación de la integración **multi-video-player-example**.

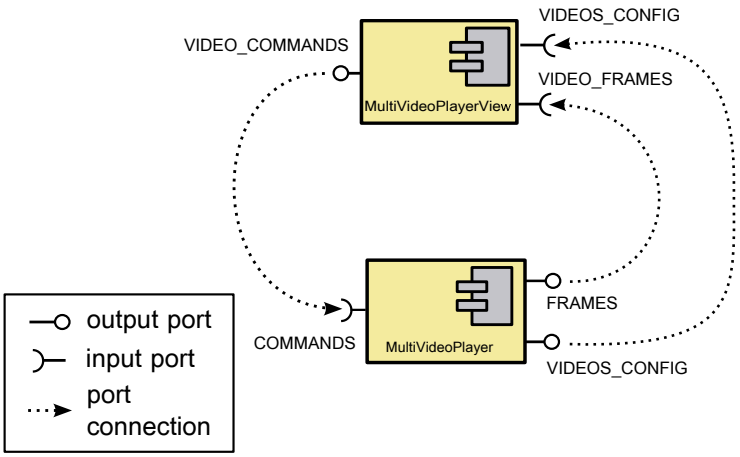

Figura 10.33: Integración multi-video-player-example

Como se observa en la figura 10.33 y en el código CoolBOT adjunto, multi-video-player-example crea un sistema interconectado entre el componente MultiVideoPlayer y la vista MultiVideoPlayerView.

Código CoolBOT que representa la integración multi-video-player-example.

```
/*
 * File: multi-video-player-example.coolbot-integration
 * Description: description file for multi-video-player-example integration.
 * Date: 20 October 2012
 * Generated by coolbot-bundle version 1.1.0
 */
integration multi-video-player-example
{
  header
  {
    author "Bycor Sanchez Sanchez <bycor.sanchez@gmail.com>";
    description "MultiVideoPlayerExample CoolBOT integration.";
    institution "IUSIANI - Universidad de Las Palmas de Gran Canaria";
    version "0.1"
  };

  machine addresses
  {
    local dis172ac: "dis172acplg.dis.ulpgc.es";
  };

  local instances
  {
    component thePlayer:MultiVideoPlayer;
    view theView:MultiVideoPlayerView with description "MultiVideoPlayer";
  };

  port connections
  {
    connect thePlayer:FRAMES to theView:VIDEO_FRAMES;
    connect thePlayer:VIDEOS_CONFIG to theView:VIDEOS_CONFIG;
    connect theView:VIDEO_COMMANDS to thePlayer:COMMANDS;
  };
};
```

Con esta integración se consigue la interacción entre la interfaz gráfica y el control del sistema de vídeo. La comunicación entre vista y componente permite que cuando el usuario seleccione un comando, éste sea transmitido al componente MultiVideoPlayer y actúe en consecuencia enviando las imágenes y datos de configuración a la vista MultiVideoPlayerView cuando correspondan.

La integración multi-video-player-example constituye la infraestructura software del sistema de vídeo distribuido.

10.3.3. Prueba de utilidad

En este apartado se pretende **demostrar la utilidad** que el **sistema de vídeo distribuido** desarrollado es capaz de aportar. Con ese fin crea un nuevo componente que aporte una funcionalidad extra al sistema, concretamente la detección de caras en las imágenes de los vídeos.

Para lograr este funcionalidad en el nuevo componente se utilizará el detector de múltiples caras ENCARA2[9]. Este detector facial ha sido desarrollado en el Instituto Universitario SIANI y en el DIS de la ULPGC.

Existe un espacio de trabajo en CoolBOT, llamado **coolbot-encara2-bundle**, el cual posee software CoolBOT relacionado con ENCARA2. En concreto el componente **Encara2** perteneciente a ese bundle detecta caras sobre una única imagen.

Este componente servirá como referencia para el posterior desarrollo de un componente que detecte caras en múltiples imágenes de diferentes vídeos.

10.3.3.1. Componente Encara2

El componente Encara2 se encuentra en el espacio de trabajo **coolbot-encara2-bundle**. Se puede ver en la figura 10.34 que dispone de un puerto de entrada llamado `IMAGE` y un puerto de salida llamado `ENCARA2_IMAGE`.

Ambos puertos trabajan con imágenes en formato RGB (red, green, blue) las cuales son transportadas en "port packets" PacketRGBImage, tipo de "port packet" definido por defecto en CoolBOT para este tipo de datos.

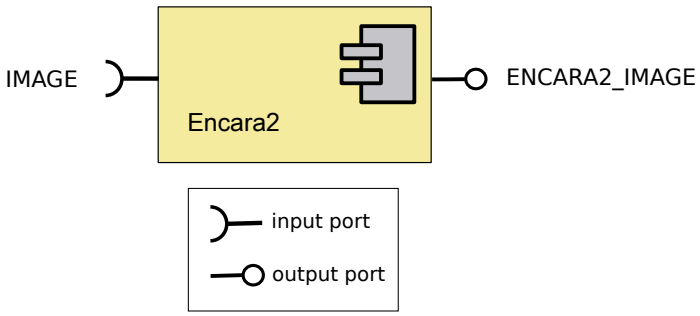

Figura 10.34: Componente Encara2

A continuación se adjunta el código CoolBOT de este componente. A partir de él es posible extraer que dispone de un estado principal llamado "Loop" el cual itera cada vez que recibe una nueva imagen, además de sus ya nombrados puertos.

Código CoolBOT que representa al componente Encara2.

```
/*
 * File: encara2.coolbot-component
 * Description: description file for Encara2 component
 * Date: 13 May 2012
 * Generated by coolbot-bundle version 1.1.0
 */
component Encara2
{
  header
  {
    author "Antonio Carlos Dominguez Brito <adominguez@iusiani.ulpgc.es>";
    description "Component Encara2 - this component applies Encara2 in a
        stream of RGB images";
    institution "IUSIANI - Universidad de Las Palmas de Gran Canaria.";
    version "0.1";
  };

  input port IMAGE type poster port packet PacketRGBImage;

  output port ENCARA2_IMAGE type poster port packet PacketRGBImage;

  entry state loop
  {
    transition on IMAGE;
  };
};
```

Internamente Encara2 aplica las funciones propias de la biblioteca ENCARA2[9] para detectar las posibles caras existentes en la imagen suministrada. En caso de detectar una o múltiples caras el componente modifica la imagen dibujando sobre ella un recuadro, en caso de no detectar ninguna la imagen permanece intacta. Finalmente, la imagen resultante tras el procesamiento es enviada por el puerto de salida `ENCARA2_IMAGE`.

10.3.3.2. Componente MultiEncara2

MultiEncara2 es un componente desarrollado para detectar la presencia de caras en múltiples imágenes. Para lograr su funcionalidad, MultiEncara2 utiliza la biblioteca de detección facial ENCARA2[9] y utiliza como referencia el componente Encara2, sección 10.3.3.1.

Al estar relacionado con el sistema de vídeo distribuido el presente componente se encuentra en el espacio de trabajo **coolbot-camera-ip-bundle**.

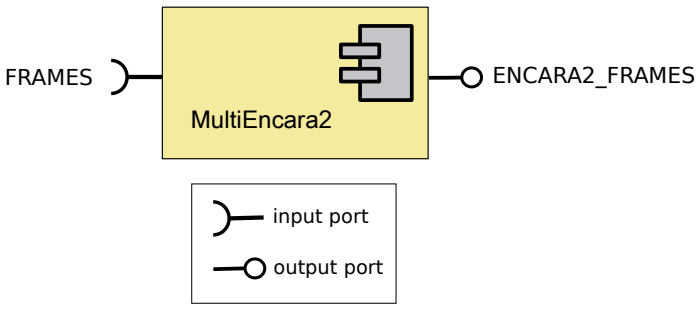

Figura 10.35: Componente MultiEncara2

A partir de su código CoolBOT, adjunto más adelante, se extraen los siguientes parámetros del componente MultiEncara2.

Existe un puerto de entrada, llamado `FRAMES`, a través del cual se recibirán las imágenes, y un puerto de salida, llamado `ENCARA2_FRAMES`, por el cual serán enviadas las imágenes tratadas. Ambos puertos utilizan para el transporte de datos el FramesPacket, tipo de "port packet" creado en el paquete MultiVideoPackets para la emisión y recepción de múltiples imágenes de manera simultánea.

El componente posee un estado llamado "Loop" con una transición asociada al puerto de entrada `FRAMES`.

Código CoolBOT que representa al componente MultiEncara2.

```
/*
 * File: multi-encara2.coolbot-component
 * Description: description file for MultiEncara2 component
 * Date: 13 November 2012
 * Generated by coolbot-bundle version 1.1.0
 */
component MultiEncara2
{
  header
  {
    author "Bycor Sanchez Sanchez <bycor.sanchez@gmail.com>";
    description "MultiVideoPlayer CoolBOT component - this component applies
        Encara2 on RGB images from multiple videos";
    institution "IUSIANI - Universidad de Las Palmas de Gran Canaria";
    version "0.1"
  };

  constants
  {
    private PACKETS_RESTART_DETECTOR=275; // Packets to treat before restarting
        face detectors
  };

  // Input ports
  input port FRAMES type poster port packet MultiVideoPackets::FramesPacket;

  // Output ports
  output port ENCARA2_FRAMES type poster port packet MultiVideoPackets::
      FramesPacket;

  entry state loop
  {
    transition on FRAMES;
  };
};
```

Definir la funcionalidad concreta del componente requiere la creación de un autómata propio. En este caso el **diseño del autómata** que detecte las caras en múltiples imágenes se compone de un estado llamado "Loop" que posee una transición "FRAMES". El autómata está representado en la figura 10.36.

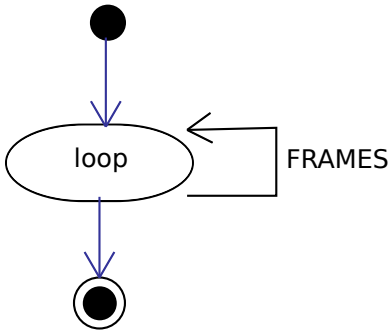

Figura 10.36: Autómata MultiEncara2

La transición del estado "Loop" está asociada al puerto de entrada `FRAMES`, de ahí que compartan nombre, y se produce cada vez que un nuevo FramesPacket es recibido en el componente. Esta transición está vinculada a la función "**loopFRAMES**".

Función loopFRAMES. Esta función se encarga de utilizar las herramientas de la biblioteca ENCARA2[9] sobre cada una de las imágenes recibidas con el fin de lograr la detección facial. Para ello sigue el siguiente esquema.

1. En primer lugar **extrae las imágenes contenidas** en el "port packet" FramesPacket, recibido por el puerto `FRAMES`.
2. Cada **imagen** es **transformada a un formato adecuado** para la biblioteca ENCARA2. Las imágenes contenidas en el FramesPacket están en formato RGB (red, green blue), sin embargo, ENCARA2 requiere imágenes en formato BGR (blue, green, red) ya que internamente utiliza la biblioteca OpenCV[28] que trabaja sobre dicho formato. Para la transformación se utiliza una función de CoolBOT que facilita el intercambio entre canales rojo y azul de una imagen.
3. En este momento **se suministra la imagen a ENCARA2** para que la procese y determine si existe alguna cara en ella. Para ello un detector utiliza la función "**ApplyENCARA2**" sobre la imagen facilitada.
4. En caso de haber detectado alguna cara tras procesar la imagen **se modifica para añadir un recuadro** sobre ella. En este caso el detector utiliza la función "**PaintFacialData**" para dibujarlo.
5. **La imagen resultante es transformada a su formato original**, es decir, de formato BGR (blue, green, red) al formato utilizado por el "port packet" FramesPacket, RGB (red, green, blue). La función que permite esta transformación es la misma utilizada en el segundo paso.
6. Después de haber tratado todas las imágenes, éstas son almacenadas en un nuevo FramesPacket para su **envío por el puerto de salida** `ENCARA2_FRAMES`. El resto de elementos del "port packet" entrante que no son imágenes mantienen sus datos originales.

La figura 10.37 muestra el funcionamiento de la función "loopFRAMES" asociada a la transición del componente MultiEncara2.

Es necesario especificar que cada imagen tratada ha de utilizar su propia instancia de ENCARA2. Esto se debe a que ENCARA2 utiliza información de imágenes anteriores para la detección facial en las actuales. Si tenemos en cuenta que cada imagen del FramesPacket corresponde a un vídeo diferente, ENCARA2 podría fallar en caso de utilizar una única instancia para todas, pues su información visual difiere.

Debido al tiempo que ENCARA2 tarda en procesar cada imagen es posible que el componente MultiEncara2 **ralentice el sistema de vídeo integrado**, especialmente en caso de realizar la detección en muchas imágenes simultáneamente.

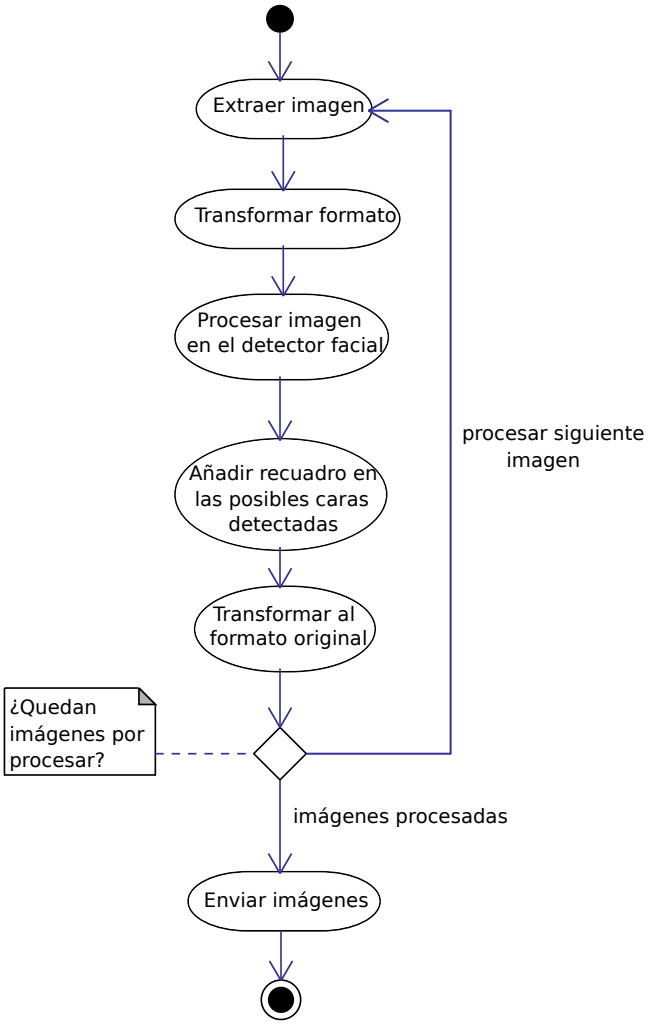

Figura 10.37: Funcionamiento loopFRAMES

10.3.3.3. Integración multi-video-player-example

Para demostrar el correcto funcionamiento de MultiEncara2 y la utilidad que aporta el sistema de vídeo distribuido desarrollado se añade a la integración multi-video-player-example el componente MultiEncara2. Esta integración está disponible en el **coolbot-camera-ip-bundle**.

La figura 10.38 representa cómo queda la integración tras añadir la funcionalidad de MultiEncara2. El sistema está formado por el componente MultiVideoPlayer, el componente MultiEncara2 y la vista MultiVideoPlayerView.

Entre las conexiones cabe destacar la existente entre el puerto de salida `FRAMES` del componente MultiVideoPlayer y el puerto de entrada `FRAMES` del componente MultiEncara2. Esta conexión posibilita que las imágenes sean tratadas para la detección de caras antes de ser mostradas en la interfaz de usuario.

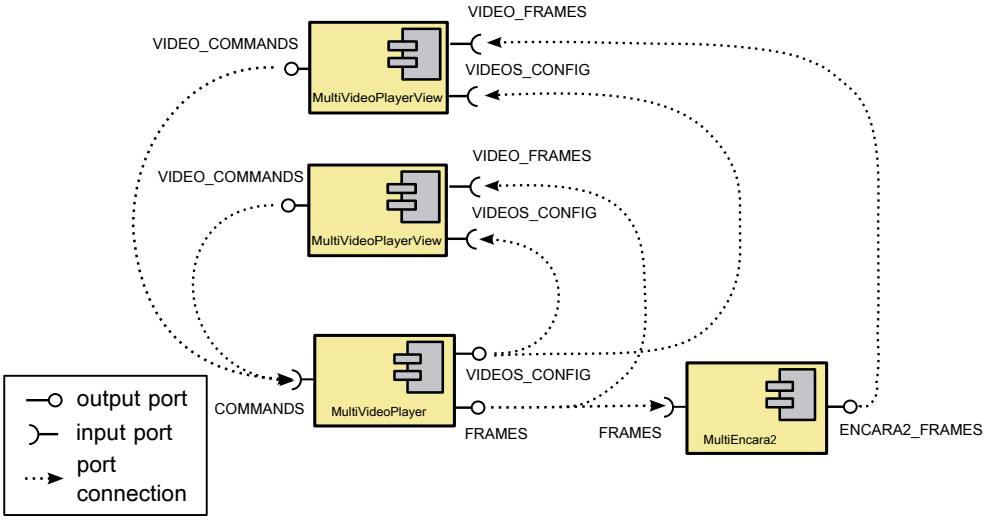

Figura 10.38: Integración multi-video-player-example

En el código CoolBOT adjunto se puede observar como la integración mantiene el sistema de vídeo original de la sección 10.3.2.4. No obstante, para la implementación del nuevo sistema con detector facial se ha creado una nueva instancia de MultiVideoPlayerView de tal forma que ambos sistemas convivan conjuntamente. Esta característica de diseño permite que ambos sistemas sean accesibles durante la ejecución.

Código CoolBOT que representa a la integración multi-video-player-example.

```
/*
 * File: multi-video-player-example.coolbot-integration
 * Description: description file for multi-video-player-example integration.
 * Date: 20 October 2012
 * Generated by coolbot-bundle version 1.1.0
 */
integration multi-video-player-example
{
  header
  {
    author "Bycor Sanchez Sanchez <bycor.sanchez@gmail.com>";
    description "MultiVideoPlayerExample CoolBOT integration.";
    institution "IUSIANI - Universidad de Las Palmas de Gran Canaria";
    version "0.1"
  };

  machine addresses
  {
    local dis172ac: "dis172acplg.dis.ulpgc.es";
  };

  local instances
  {
    component thePlayer:MultiVideoPlayer;
    component theEncara2:MultiEncara2;

    view theView:MultiVideoPlayerView with description "MultiVideoPlayer";
    view theEncara2View:MultiVideoPlayerView with description "Encara2
        MultiVideoPlayer";
  };
```

```
  port connections
  {
    connect thePlayer:FRAMES to theView:VIDEO_FRAMES;
    connect thePlayer:VIDEOS_CONFIG to theView:VIDEOS_CONFIG;
    connect theView:VIDEO_COMMANDS to thePlayer:COMMANDS;

    connect thePlayer:FRAMES to theEncara2:FRAMES;
    connect theEncara2:ENCARA2_FRAMES to theEncara2View:VIDEO_FRAMES;
    connect thePlayer:VIDEOS_CONFIG to theEncara2View:VIDEOS_CONFIG;
    connect theEncara2View:VIDEO_COMMANDS to thePlayer:COMMANDS;
  };
};
```

Añadido este sistema a la integración multi-video-player-example se consigue aportar una funcionalidad específica, como es la detección facial, al sistema de vídeo distribuido desarrollado en etapas anteriores del presente TFG.

En la figura 10.39 se muestra el funcionamiento del sistema de vídeo haciendo uso del componente MultiEncara2.

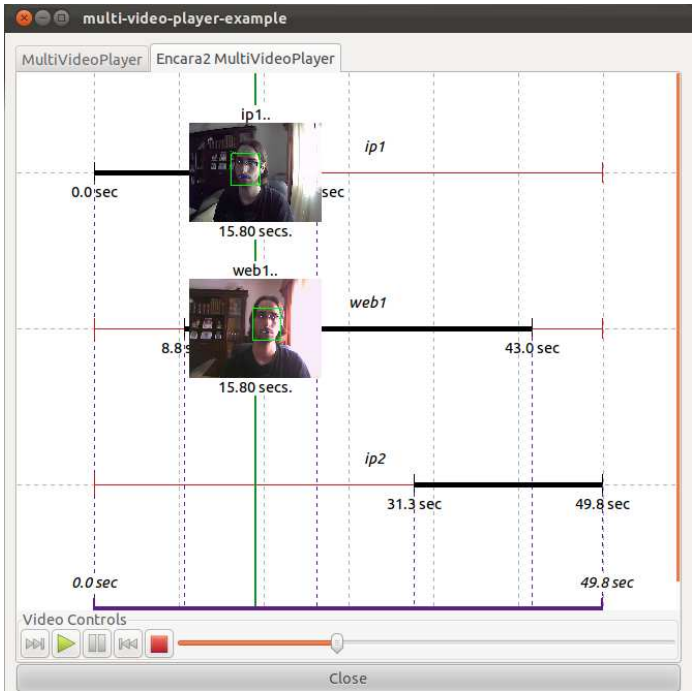

Figura 10.39: Funcionamiento de multi-video-player-example con MultiEncara2

Como se puede observar durante la reproducción en el sistema de vídeo integrado las caras detectadas en las imágenes de los vídeos son destacadas colocando a su alrededor un cuadrado de color verde.

La integración multi-video-player-example con el detector facial demuestra las utilidades que el sistema de vídeo distribuido es capaz de aportar. Adicionalmente se comprueba el correcto funcionamiento del componente MultiEncara2 creado.

Capítulo 11

Pruebas

A continuación se realizan pruebas que demuestran el correcto funcionamiento del material desarrollado a lo largo del trabajo. Este capítulo se estructura siguiendo las tres principales etapas: integración de cámaras IP, almacenamiento en vídeo y sistema de vídeo distribuido.

11.1. Integración de cámaras IP

En este apartado se comprueba que la infraestructura software creada para la integración de cámaras IP funcione de forma correcta. Para ello es necesario comprobar que el componente CameraIPAxis, diseñado y desarrollado para dicho propósito, funciona bien.

Durante el desarrollo de la integración de cámaras IP fue creada una integración llamada **camera-ip-axis-example**. Dicha integración, conformada por el componente CameraIPAxis y una vista, permite visualizar en tiempo real las imágenes que envía la cámara IP. La implementación de camera-ip-axis-example se ve reflejada en la sección 10.1.3.2.

Ejecutar camera-ip-axis-example permitirá comprobar si se ha logrado una correcta integración de la cámara de red. Para una correcta conexión la integración necesita los siguientes parámetros: IP o dominio de la cámara de red, usuario de autentificación y contraseña correspondiente al usuario indicado.

El primer paso a realizar es verificar que los posibles **errores de conexión** con la cámara IP **son controlados**. Para ello se realizan dos pruebas de ejemplo en las que se aportan datos erróneos.

En la primera prueba se indica una dirección IP errónea del dispositivo. Como se puede observar en la figura 11.1 se produce una excepción tipo "connection_error" que informa de que el intento de conexión con la cámara IP falló. Además, con el

mensaje "wrong hostname or host down" especifica que la dirección o dominio de red suministrado no es correcto o el dispositivo no está encendido.

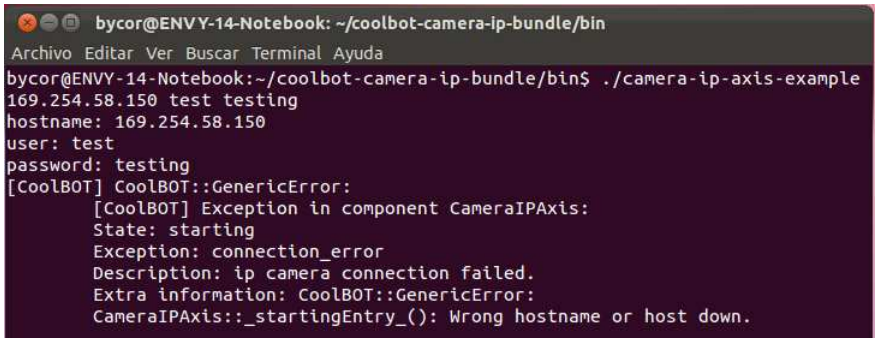

Figura 11.1: Dirección IP suministrada errónea.

En la segunda prueba aportan datos de autentificación erróneos. La figura 11.2 muestra como ante este fallo de conexión se produce nuevamente una excepción del tipo "connection_error".

El mensaje "unauthorized or wrong login", incluido como dato final de la excepción, informa de que los datos de autentificación suministrados son inválidos o bien pertenecen a un usuario que no tiene los permisos necesarios para acceder a la cámara IP.

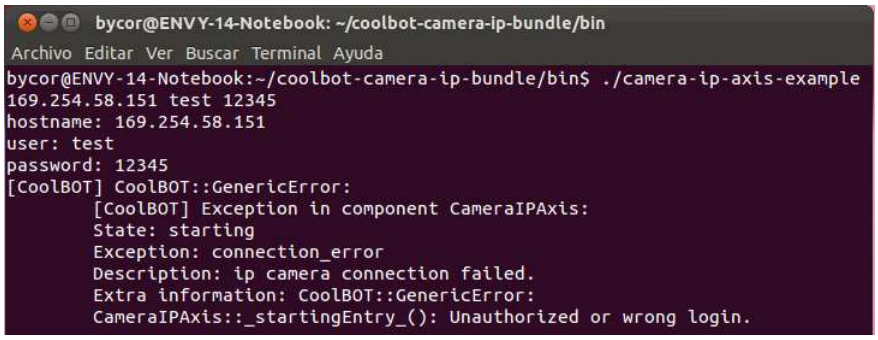

Figura 11.2: Datos de autentificación erróneos

El siguiente paso es realizar una prueba suministrando los parámetros correctamente. Como se puede ver en la figura 11.3 tras conectarse al dispositivo la integración **muestra por pantalla las imágenes** obtenidas a través de la red **de la cámara IP**.

Con esta prueba final se puede concluir que se ha conseguido el objetivo inicial de desarrollar una infraestructura software para la reproducción de vídeo remota de cámaras IP Axis.

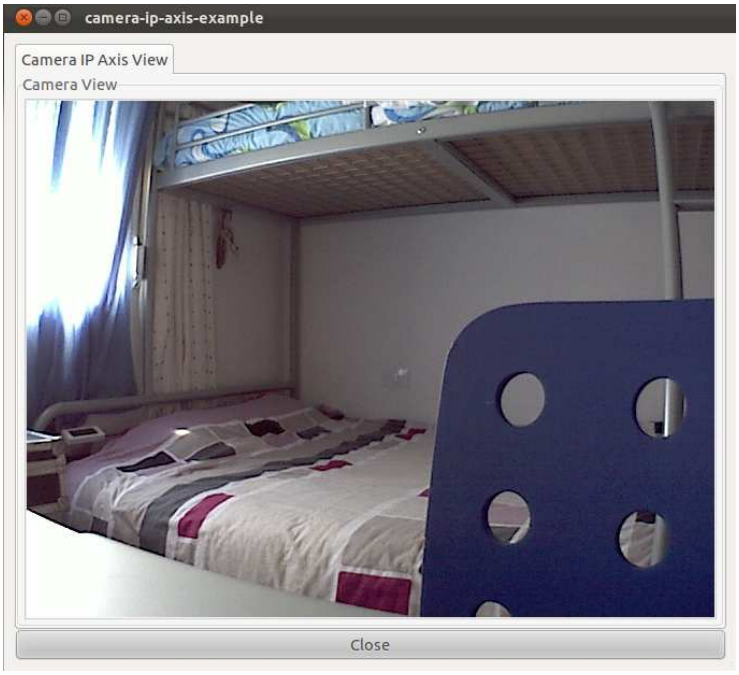

Figura 11.3: Prueba integración cámaras IP

11.2. Almacenamiento en vídeo

En este apartado se comprueba que la infraestructura software creada para el almacenamiento, gestión y acceso distribuido de vídeo funciona y muestra ejemplos gráficos haciendo uso de ella.

Durante la etapa de desarrollo de almacenamiento en vídeo se ha creado una integración llamada **camera-ip-axis-recorder**, sección 10.2.2. Dicha integración está compuesta por el componente CameraIPAxis y el componente VideoRecorder, entre los cuales permiten la grabación en vídeo de las imágenes de la cámara de red, camera-ip-axis-recorder también utiliza una vista para ver las imágenes que están siendo grabadas y además decidir cuándo finalizar la grabación.

Con motivo de corroborar el correcto funcionamiento del **almacenamiento en vídeo** se hará una prueba que ejecute esta integración. Para realizar una correcta conexión es necesario suministrar los parámetros: IP o dominio de la cámara de red, usuario de autentificación y contraseña correspondiente al usuario indicado. Adicionalmente también hay que aportar el nombre del archivo de vídeo generado.

Durante la ejecución son visibles las imágenes de la cámara IP que, simultáneamente, van siendo almacenadas. Una vez finalizada la ejecución se habrá generado un archivo de vídeo con las imágenes del dispositivo de red.

Para demostrar que las imágenes vistas durante la ejecución han sido almacenadas en el archivo de vídeo se procede a su reproducción.

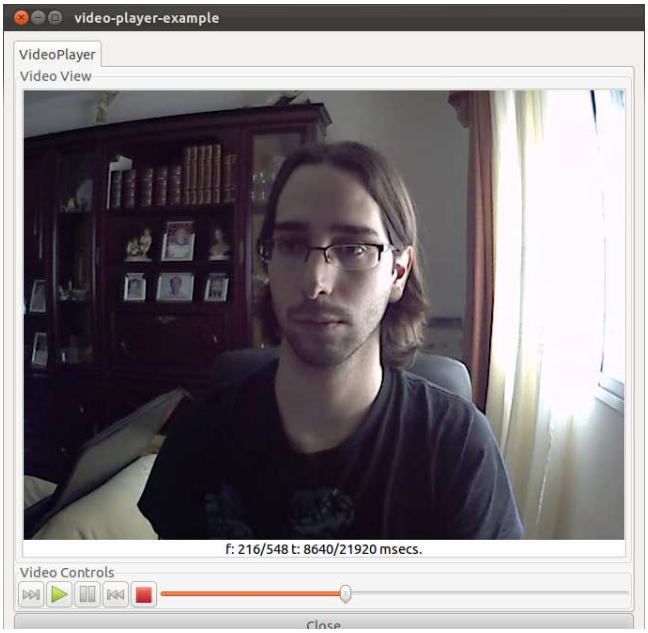

Figura 11.4: Reproducción vídeo

La figura 11.4 corresponde a la reproducción de dicho vídeo mediante la integración video-player-example, utilizada en la sección 10.3.1.4 como referencia para el sistema de vídeo distribuido.

Observada esta prueba es posible concluir que la infraestructura software para el almacenamiento de vídeo distribuido funciona bien. Así el segundo objetivo propuesto en el presente trabajo se considera cumplido.

11.3. Sistema de vídeo distribuido

En este último apartado se pretende demostrar que el sistema de gestión de vídeo implementado cumple su funcionalidad, así como se muestran ejemplos gráficos del mismo.

A lo largo del desarrollo de este sistema de gestión se creó el componente MultiVideoPlayer y la vista MultiVideoPlayerView. Ambos son combinados en la integración **multi-video-player-example** para formar el sistema de distribuido de vídeo final.

Este sistema distribuido ofrece un **acceso integrado** a las secuencias de los **vídeos grabados** durante la anterior etapa del trabajo, almacenamiento de imágenes procedentes de cámaras IP en vídeo. Además, permite reproducir múltiples vídeos de manera simultánea y la reproducción de un único vídeo en pantalla completa.

Para su ejecución la integración necesita como parámetros la ruta donde se encuentran los datos del software ENCARA2[9] y los diferentes vídeos que conformarán el sistema.

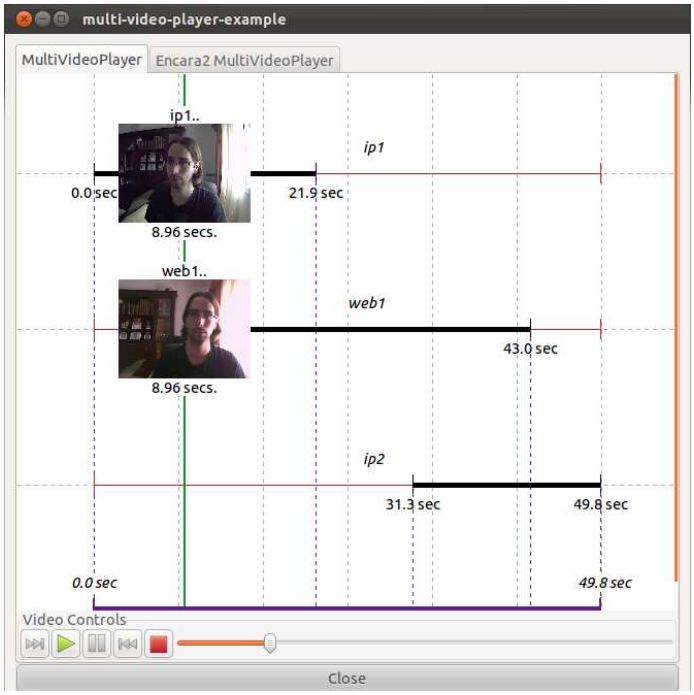

Figura 11.5: Ejemplo sistema de vídeo distribuido

El sistema también permite la reproducción en pantalla completa de un único vídeo, pulsando el botón izquierdo del ratón dos veces sobre su imagen. Este modo de reproducción facilita una mejor visión del vídeo seleccionado.

Con esta prueba es posible concluir que el último de los objetivos del trabajo se ha cumplido de forma satisfactoria.

11.4. Sistema con detección facial

Con el objetivo de demostrar las utilidades que esta infraestructura es capaz de aportar se ha incluido a la integración **multi-video-player-example** el componente MultiEncara2. Este componente aplica las funciones del software ENCARA2[9] para la detección de caras en las imágenes del sistema distribuido de vídeo.

Para comprobar su funcionamiento se procede a la ejecución de multi-video-player-example suministrando los mismos parámetros que en el apartado anterior: ruta donde se encuentran los datos del software ENCARA2[9] y los nombres de los vídeos que conforman el sistema.

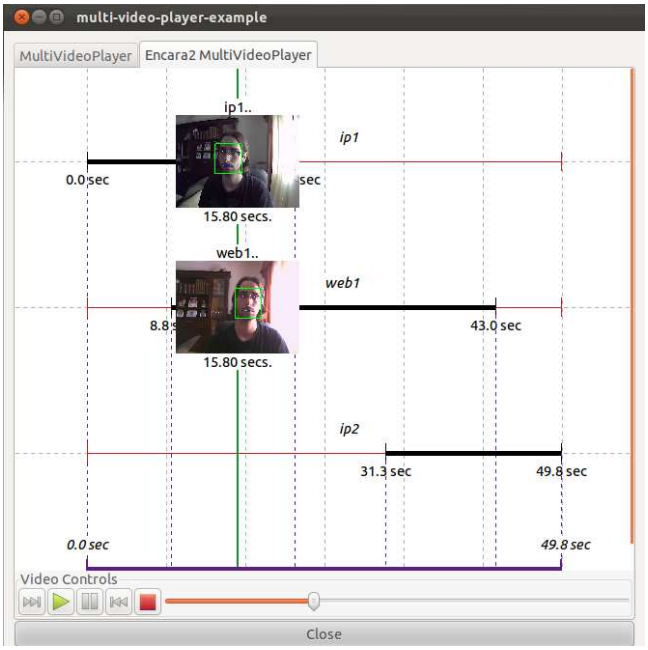

Figura 11.6: Sistema de vídeo distribuido con detección facial

La figura 11.6 muestra como el sistema es capaz de detectar caras en múltiples imágenes, para lo cual añade un recuadro de color verde alrededor de cada cara localizada. Es importante matizar que el tratamiento de las imágenes por parte de ENCARA2 requiere cierto tiempo, lo que puede afectar a la fluidez de la reproducción.

Esta prueba confirma que es posible añadir nuevas funciones al sistema de vídeo distribuido que lo convierten en una herramienta operativa para tratar múltiples vídeos.

Capítulo 12

Conclusiones

A continuación se esgrimen las conclusiones del presente Trabajo Fin de Grado.

12.1. Conclusiones

El objetivo de este Trabajo Fin de Grado fue el desarrollo de un sistema de gestión de vídeo distribuido utilizando cámaras IP, creando para ello software que integrara estos dispositivos, permitiera el almacenamiento de sus imágenes y finalmente crear un sistema de vídeo distribuido.

Desde el punto de vista del objetivo inicial los resultados obtenidos son satisfactorios. Se ha conseguido desarrollar un sistema de gestión capaz de monitorizar una red compuesta por cámaras IP.

Durante el desarrollo del trabajo se han extraído las siguientes conclusiones:

- La comunicación con cámaras IP requiere de un profundo **estudio previo** de su interfaz de programación (API) a fin de descubrir sus peculiaridades y comportamiento como por ejemplo: qué protocolos soporta, cómo se comunica, en que formato suministra los datos, etc.

- Es esencial la elección, estudio y familiarización de **herramientas software** que ayuden en la comunicación vía HTTP y tratamiento de los datos con la cámaras de red.

- La recepción de los datos de cámaras IP está profundamente condicionado por parámetros como el ancho de banda disponible, nivel de compresión de las imágenes, resolución de la imagen, entre otros. Estos elementos pueden provocar que el software que integra estos dispositivos no reciba imágenes con la frecuencia deseada y, por tanto, la fluidez de la reproducción se vea afectada.

- La reproducción simultánea de múltiples vídeos en el sistema distribuido de vídeo puede repercutir en una pérdida de fluidez en el sistema. El motivo es el coste que supone la captación y envío de muchas imágenes para ser visualizadas.
- Se ha demostrado la **utilidad** que el sistema de gestión de vídeo distribuido es capaz de aportar en una red de vigilancia constituida por cámaras IP. Además, su diseño permite la creación de software para el tratamiento de imágenes, como se ha demostrado en el desarrollo del detector facial.

Todo **el material desarrollado** a lo largo del trabajo se encuentra **disponible en la página web del proyecto CoolBOT**[6] para su descarga en el "bundle" **coolbot-camera-ip-bundle**.

Asimismo, se consideran alcanzados de forma exitosa los objetivos académicos planteados, utilizado conocimiento de entre otras asignaturas:

- Algoritmos, programación y estructuras de datos
- Sistemas empotrados y de tiempo real
- Algoritmos y programación paralela
- Sistemas operativos
- Interfaces humanas
- Redes

12.2. Trabajo futuro

A pesar de considerar el material desarrollado como muy satisfactorio siempre es posible realizar mejoras sobre él. Además, a partir de este trabajo pueden surgir nuevos proyectos relacionados con cámaras IP. De esta forma se proponen las siguientes mejoras y posibles líneas de desarrollo:

- Dar soporte a cámaras IP de otros fabricantes existentes en el mercado, más allá de las pertenecientes a la empresa Axis Communications[3].
- Posibilitar la **captación de audio**.
- Incluir **controles para el movimiento, zoom y enfoque** de los modelos de cámaras IP que así lo permitan de forma remota (no manual).
- Mejorar la transmisión de datos entre la cámara IP y el equipo utilizando un formato de imagen más eficiente. De esta manera, al utilizar un menor ancho de banda, las imágenes serán enviadas con mayor fluidez.
- Modificar el almacenamiento en vídeo de las imágenes para realizar una **grabación en tiempo real**, de tal forma que se genere un archivo de vídeo reproducible al mismo tiempo que continúa la grabación. Así se evitaría la necesidad de parar la grabación para visualizar las secuencias de vídeo.
- Creación de software para el **tratamiento de imágenes** procedentes de estos dispositivos como, por ejemplo, la detección de matrículas. Ya se ha mostrado esta posibilidad en el presente trabajo con la detección de caras.

Apéndice A

Manual de Usuario

En este capítulo se va a explicar cómo utilizar el hardware y software empleado a lo largo de este Trabajo de Fin de Grado.

A.1. Instalación de cámara IP

Siguiendo las instrucciones del fabricante los pasos para la instalar la cámara de red son los siguientes.

A.1.1. Instalación del hardware

1. Conecte la cámara a la red con un cable de red blindado. Esta conexión es temporal y permite configurar los valores de la cámara mediante la red por cable antes de la conexión a la red inalámbrica.

2. Conecte la alimentación utilizando uno de los métodos especificados en la lista que aparece a continuación:

   - El conector de alimentación suministrado.
   - PoE (corriente a través de Ethernet). Si está disponible para el modelo de cámara, se detecta automáticamente cuando el cable de red está conectado
   - Conecte la alimentación mediante el conector de terminal (una conexión especial situada en la parte posterior del dispositivo).

3. Compruebe que los indicadores LED anuncian las condiciones adecuadas, conectado a la corriente y conectado a la red fija o inalámbrica. Tenga en cuenta que algunos LED pueden deshabilitarse y apagarse.

A.1.2. Configuración de dirección IP

Actualmente, la mayoría de redes disponen de un servidor DHCP que asigna direcciones IP a dispositivos conectados de forma automática. En caso de no disponer de un servidor DHCP, las cámaras de red AXIS utilizarán 192.168.0.90 como dirección IP predeterminada.

Si desea configurar la dirección en entorno Unix/Linux/Mac el método recomendado a utilizar es ARP/Ping. Para ello:

1. Adquiera una dirección IP en el mismo segmento de red en el que está conectado su equipo.

2. Localice el número de serie (N/S) en la etiqueta de la cámara AXIS (parte inferior del dispositivo).

3. Abra una línea de comando en su equipo y escriba los comandos siguientes:

```
arp -s <Direccion IP> <Numero de serie> temp
ping -s 408 <Direccion IP>
```

   Ejemplo de uso:

```
arp -s 192.168.0.125 00:40:8c:18:10:00 temp
ping -s 408 192.168.0.125
```

4. Compruebe que el cable de red está conectado a la cámara y desconecte y vuelva a conectar la corriente para iniciarla o reiniciarla.

5. Cierre la línea de comandos cuando vea 'Reply from 192.168.0.125: ...' o similar.

6. Acceda por el explorador, escribiendo http://<dirección IP> en el campo Ubicación/Dirección.

Otros métodos para obtener o configurar la IP de la cámara de red son: AXIS Dynamic DNS Service, servicio gratuito online que concede Axis para sus productos, o la visualización de las páginas admin. del servidor DHCP, en las cuales aparecerá la IP asignada actualmente.

A.1.3. Configuración de conexión inalámbrica

Una vez conectada la cámara IP AXIS a la red, pueden configurarse los valores de la conexión inalámbrica. El método más rápido y seguro para configurarla se basa en el uso de una conexión por cable, ya que se deshabilita la conexión inalámbrica y se garantiza una confidencialidad mayor durante la introducción de los valores.

Por lo general los dispositivos AXIS buscan de forma automática las conexiones de red disponibles y sólo permite que esté activa una cada vez. No se recomienda configurar la cámara con una conexión inalámbrica no segura.

Para la configuración de los datos de la conexión inalámbrica es necesario acceder al apartado Setup > Basic Configuration > Wireless.

En dicho apartado es posible configurar los valores inalámbricos seleccionando una de las redes inalámbricas que sean captadas en ese instante por la cámara IP o mediante la creación de una nueva. Entre los valores a escoger están los métodos de seguridad, de forma general los tipos admitidos son: WPA-/WPA2-PSK, WPA-/WPA2-Enterprise y WEP.

Una vez finalizada la configuración inalámbrica desconecte el cable de red de la cámara y tras 20 o 30 segundos actualice el navegador para confirmar la conexión inalámbrica. En caso de haberse equivocado con los datos y no se conecte la cámara a la red siempre es posible restablecer los valores predeterminados de fábrica.

A.2. Instalación del software requerido

En esta sección se especifican los pasos a seguir para instalar el software requerido que permita la correcta ejecución del material desarrollado. En muchos de los casos estas instrucciones están detalladas en el archivo INSTALL que encontraremos en el material descargado. Especificar además que las pautas de instalación nombradas a continuación son las utilizadas para un sistema Linux normal, como el utilizado para la realización de este proyecto.

A.2.1. Libcurl

A.2.1.1. Requisitos

En este caso libcurl no especifica requisitos previos para su instalación, excepto OpenSSL en caso de querer hacer uso del protocolo https.

A.2.1.2. Instalación

La librería libcurl[21] forma parte de cURL, una herramienta de línea de comandos la cual soporta una infinidad de protocolos de red. Es por ello que para su instalación procederemos a instalar cURL.

1. Decargar la última versión estable de cURL desde su página web oficial[21] para la distribución de Linux requerida.
2. Extraer los archivos fuente del documento comprimido que se ha descargado con anterioridad.
3. Acceder desde un terminal a la carpeta extraída que contiene los archivos fuente y configurarlos. Para ello realizar los siguientes comandos.

```
cd <Ruta carpeta>
./configure
```

4. Compilar el código utilizando el comando:

```
./make
```

5. Finalmente, si no han ocurrido errores en etapas anteriores, instalamos libcurl con el comando:

```
./make install
```

   Este último paso requiere permisos de administrador.

Tras la realización de todos los pasos libcurl estará instalado en nuestro equipo.

A.2.2. OpenCV

A.2.2.1. Requisitos

Para la correcta instalación de OpenCV[28] se requiere instalar previamente las librerías y herramientas de las cuales depende. Algunas de esas dependencias son opcionales, es decir, sólo es necesaria su instalación para usos concretos. Para el uso que en este trabajo se da a OpenCV habrá que instalar: compilador de C/C++, cmake 2.6 o mayor, pkg-config, ffmpeg, libgstreamer, libv4l, libxine, unicap, libdc1394 2.x.

También es recomendable tener instalado las librerías: libjpeg, libtiff, libjasper, libpng and zlib, openexr.

Dichos requisitos son susceptibles de ser alterados, por lo que es recomendable consultarlos en la página oficial de OpenCV[28].

A.2.2.2. Instalación

1. Descargar la última versión estable disponible de OpenCV desde su página web oficial[28] para la distribución requerida.
2. Extraer los archivos fuente del documento descargado en el paso anterior.
3. Crear una nueva carpeta en el interior de la carpeta extraída. La carpeta creada contendrá makefile que genera cmake.
4. Acceder desde un terminal al interior de la nueva carpeta del paso anterior y ejecutar la herramienta cmake con el comando:

```
cmake -D CMAKE\_BUILD\_TYPE=RELEASE -D CMAKE\_INSTALL\_PREFIX=/usr/
    local ..
```

5. Compilar haciendo uso del comando:

```
./make
```

6. Instalar OpenCV con el comando:

```
./make install
```

   Este último paso requiere permisos de administrador.

Una vez finalizados todos los pasos OpenCV estará instalado en nuestro equipo. En caso de error en las etapas de compilación comprobar que las librerías nombradas en el apartado de requisitos estén instaladas.

A.2.3. CoolBOT

A.2.3.1. Requisitos

Las librerías y herramientas que requiere CoolBOT[6] para su correcta instalación son: Ace, GTK+ 2.0 o mayor, Bison, Flex, pkg-config, cmake y git. Git no es imprescindible para la instalación, pero sí para la descarga.

A.2.3.2. Instalación

1. Situarse en el directorio donde se quiera instalar CoolBOT y descargarlo utilizando el software de control de versiones git. La orden para ello es:

```
git clone git://coolbotproject.dis.ulpgc.es/coolbot-project
```

2. Acceder a la carpeta "build" que se encuentra en el interior del directorio descargado.

```
cd coolbot-project/build
```

3. Ejecutar el comando

```
ccmake ..
```

   Entre las opciones que aparecen en pantalla deshabilitar el modo depuración, poniendo DEBUG_MODE a OFF.

4. Pulsar 'c' para continuar y luego 'g' para generar el proyecto y salir.

5. Compilación del proyecto mediante la orden:

```
./make
```

6. Instalar CoolBOT con el comando:

```
./make install
```

   La instalación requiere permisos de administrador.

Ante cualquier error en etapas de compilación comprobar que los requisitos para la instalación se cumplen.

A.2.4. ENCARA2

A.2.4.1. Requisitos

ENCARA2[9] requiere tener instalada la última versión estable de OpenCV.

A.2.4.2. Instalación

1. Descargar la última versión disponible del proyecto ENCARA2 para Linux desde su página web[9].
2. Extraer el contenido del archivo descargado.
3. Acceder desde el terminal a la carpeta extraída y generar los archivos "makefile". Para ello utilizar los comandos:

```
cd <Ruta carpeta>
make -f makeMakeObjects.mak
```

4. Generar la biblioteca ENCARA2 mediante el comando:

```
make -f encara2-lib.mak
```

5. Generar aplicación de ejemplo con la orden:

```
make -f encara2-test.mak
```

6. Ejecutar la aplicación de ejemplo para comprobar el correcto funcionamiento de ENCARA2:

```
./bin/encara2-test -path ``<Ruta de ENCARA2>/ENCARA2data``
```

7. Añadir la ruta en la que se encuentra la librería de ENCARA2, generada en pasos anteriores, a la variable de ambiente LD_LIBRARY_PATH. Esto permitirá a las aplicaciones encontrar el archivo "libencara-raw.so" sin mayor problema.

A.3. Instalación de bundles

Para una mejor estructuración CoolBOT organiza los componentes, vistas, paquetes e integraciones en los denominados "bundles". Cada bundle contiene en su interior las unidades desarrolladas para un fin en común. Esta estructuración permite encontrar componentes previamente creados, así como facilita la compilación y ejecución de todos los elementos que contiene.

Concretamente para este Trabajo Fin de Grado se han utilizado dos bundles:

- **Coolbot-camera-bundle**: contiene componentes y vistas relacionados con *cámaras web* o *webcams*. En este bundle se encuentran componentes, vistas y paquetes utilizados principalmente como referencia en el desarrollo del trabajo. Algunos de sus elementos también han sido utilizados para aprovechar funcionalidades ya existentes como, por ejemplo, el componente VideoRecorder empleado en la grabación de vídeo distribuido, sección 10.2.

- **Coolbot-camera-ip-bundle**: bundle **desarrollado íntegramente en este TFG**. Contiene componentes, vistas, paquetes e integraciones implementadas a lo largo de la etapa de desarrollo. Todo el software de este bundle está relacionado con las cámaras IP y la gestión de vídeo distribuido. Posee elementos como el componente CameraIPAxis, creado para la integrar cámaras IP Axis, o la vista MultiVideoPlayerView, creada como interfaz gráfica para la gestión de vídeo distribuido.

Ambos bundles se encuentran disponibles para su descarga en la web del proyecto CoolBOT[6]. A continuación se detallan los requisitos y pasos a dar para la correcta instalación de ambos bundles.

A.3.1. Coolbot-camera-bundle

A.3.1.1. Requisitos

Es necesario disponer de las siguientes herramientas instaladas: CoolBOT, OpenCV y ffmpeg.

A.3.1.2. Instalación

1. Situarse en el directorio donde se quiera instalar y descargar el bundle utilizando el software de control de versiones git. La orden a utilizar para ello es:

```
git clone git://coolbotproject.dis.ulpgc.es/coolbot-camera-bundle
```

2. Acceder a la carpeta "build" situada dentro del directorio descargado:

```
cd build
```

3. Configurar los parámetros de configuración con ccmake:

```
ccmake ..
```

4. Activamos la creación de variables de entorno poniendo CREATE_ENV_VAR a ON.

5. Desactivamos el modo depuración poniendo a OFF la opción DEBUG_MODE.

6. Pulsamos 'c' para continuar y, si todo ha ido bien, pulsamos 'g' para generar el proyecto y salir (en caso de que algún requisito previo no se cumpla, CMake aborta indicando cuál de ellos no ha sido encontrado en el sistema).

7. Reiniciar el shell o interprete de comandos para aplicar los cambios. Utilizando para ello el comando:

```
source ~/.bashrc
```

8. Compilar el proyecto con el comando:

```
make
```

Estando situados en el directorio "build".

9. Instalar mediante el comando:

```
make install
```

Son necesarios permisos de administrador para la instalación.

A.3.2. Coolbot-camera-ip-bundle

A.3.2.1. Requisitos

Para la correcta instalación se requieren las siguientes herramientas instaladas en el equipo: libcurl, CoolBOT, OpenCV, coolbot-camera-bundle y ENCARA2.

A.3.2.2. Instalación

1. Situarse en el directorio donde se quiera instalar y descargar el bundle utilizando el software de control de versiones git. La orden a utilizar para ello es:

```
git clone git://coolbotproject.dis.ulpgc.es/coolbot-camera-ip-bundle
```

2. Accedemos a la carpeta "build" situada dentro del directorio descargado.

```
cd build
```

3. Configurar los parámetros de configuración con ccmake.

```
ccmake ..
```

4. Activamos la creación de variables de entorno poniendo CREATE_ENV_VAR a ON.
5. Desactivamos el modo depuración poniendo a OFF la opción DEBUG_MODE.
6. Pulsamos 'c' para continuar. Si todo ha ido bien pulsamos 'g' para generar el proyecto y salir (en caso de que algún requisito previo no se cumpla, CMake aborta indicando cuál de ellos no ha sido encontrado en el sistema).
7. Reiniciar el shell o interprete de comandos para aplicar los cambios. Utilizando para ello el comando:

```
~/.bashrc
```

8. Compilar el proyecto con el comando:

```
./make
```

Estando situados en el directorio "build".

9. Instalar mediante el comando:

```
./make install
```

La instalación requiere permisos de administrador.

A.4. Ejecución

Es imprescindible la instalación de todas las herramientas software requeridas para proceder a la ejecución. Las instalaciones están descritas en los apartados A.2 y A.3 anteriores.

Si la instalación y compilación de los bundles se han producido sin errores los archivos ejecutables se habrán generado en "/<ruta>/coolbot-camera-ip-bundle/bin". "Ruta" corresponde a la dirección donde haya sido instalado coolbot-camera-ip-bundle.

A continuación se explica cómo ejecutar las integraciones desarrolladas a lo largo de las tres principales etapas de este trabajo: integración de cámaras IP, almacenamiento en vídeo y sistema de vídeo distribuido.

A.4.1. Integración de cámaras IP

Durante el desarrollo de una infraestructura software que permita la integración de cámaras IP se creó una integración llamada camera-ip-axis-example, sección 10.1.3.2. Esta integración permite visualizar en tiempo real las imágenes enviadas por la cámara IP.

El comando utilizado para su ejecución es el siguiente.

```
./camera-ip-axis-example <hostname> <user> <password>
```

El parámetro <hostname> se ha de sustituir por la IP o nombre de dominio de la cámara de red, <user> por el usuario de autentificación y <password> por la contraseña de autentificación para el usuario indicado.

Un ejemplo del comando habiendo sustituido esos parámetros es:

```
./camera-ip-axis-example 192.168.0.90 root toor
```

A pesar de ser visibles en el terminal, los datos de autentificación van cifrados por la red.

A.4.2. Grabación cámara IP

Durante el desarrollo de la etapa de almacenamiento en vídeo fue creada una integración con el nombre de camera-ip-axis-recorder, sección 10.2.2. Esta integración permite grabar en un archivo de vídeo las imágenes emitidas por la cámara IP.

Para su ejecución se utiliza el siguiente comando.

```
./camera-ip-axis-recorder <hostname> <user> <password> <avi-file>
```

El parámetro <hostname> ha de ser sustituido por la IP o nombre de dominio de la cámara de red, <user> por el usuario de autentificación, <password> por la contraseña de autentificación para el usuario indicado y <avi-file> por un nombre

para el archivo de vídeo que se generará.

Un ejemplo del comando es el siguiente.

```
./camera-ip-axis-example 192.168.0.90 root toor grabacion1
```

A pesar de ser visibles en el terminal, los datos de autentificación van cifrados por la red.

A.4.3. Sistema de gestión de vídeo offline

Durante el desarrollo del sistema de vídeo distribuido se creó una integración llamada multi-video-player-example, sección 10.3.2.4. Dicha integración permite la gestión de numerosos vídeos introducidos en el sistema. Además, con posterioridad se añadió un nuevo sistema a multi-video-player-example para ofrecer la funcionalidad de detectar caras, sección 10.3.3.3.

La ejecución de esta integración requiere del siguiente comando.

```
./multi-video-player-example <encara2-data-path> <avi-file 1> <avi-file 2>
    ... <avi-file MAX>
```

Se ha de sustituir <encara2-data-path> por la ruta en la que se encuentra los datos de ENCARA2 ("/<Ruta-instalación-ENCARA2>/ENCARA2data") y los restantes parámetros <avi-file> por nombres de archivos de vídeo generados en la etapa de almacenamiento en vídeo.

Comando de ejemplo.

```
./multi-video-player-example /encara/v2.11/linux/ENCARA2data/ grabacion1
    -001352658018691.avi grabacion2-001352658073670.avi
```

Bibliografía

[1] ArgoUML, página oficial (inglés). http://argouml.tigris.org/.

[2] Audio Video Interleave (AVI), referencia (inglés). http://msdn.microsoft.com/en-us/library/ms779636.aspx.

[3] Axis Communications, página oficial (inglés). http://www.axis.com.

[4] CMake, página oficial (inglés). http://www.cmake.org/.

[5] Common Gateway Interface (CGI) (inglés). http://www.w3.org/CGI/.

[6] CoolBOT, página oficial del proyecto CoolBOT (inglés). http://www.coolbotproject.org/.

[7] Digest access authentication, información relativa (inglés). http://tools.ietf.org/html/rfc2617.

[8] Directiva 95/46/CE del Parlamento Europeo. http://europa.eu/legislation_summaries/information_society/data_protection/l14012_es.htm.

[9] ENCARA2, página oficial (inglés). http://new-mozart.dis.ulpgc.es/encara2.

[10] Función hash, información relativa (inglés). http://www.partow.net/programming/hashfunctions/index.html.

[11] g++, GNU compiler collection (inglés). http://gcc.gnu.org/.

[12] GIT, página oficial (inglés). http://git-scm.com/.

[13] GTK+, página oficial (inglés). http://www.gtk.org/.

[14] Hypertext Transfer Protocol (HTTP), referencia (inglés). http://www.w3.org/Protocols/HTTP/AsImplemented.html.

[15] Hypertext Transfer Protocol Secure (HTTPS) (inglés). https://tools.ietf.org/html/rfc2818.

[16] Inkscape, página oficial (inglés). http://inkscape.org/.

[17] Internet protocol (IP) (inglés). http://tools.ietf.org/html/rfc791.

[18] JPEG, información relativa (inglés). http://www.w3.org/Graphics/JPEG/.

[19] Kdevelop, página oficial (inglés). http://kdevelop.org/.

[20] Kile, página oficial (inglés). http://kile.sourceforge.net/.

[21] Libcurl, página oficial (inglés). http://curl.haxx.se/libcurl/.

[22] Licencia BSD, Berkeley Software Distribution. Definición de licencia (inglés). http://www.linfo.org/bsdlicense.html.

[23] Licencia GNU GPL, General Public License (inglés). http://www.gnu.org/copyleft/gpl.html.

[24] Licencia MIT, información relativa (inglés). http://opensource.org/licenses/MIT.

[25] Licencias GNU, página oficial (inglés). http://www.gnu.org/licenses/licenses.html.

[26] Motion-JPEG (inglés). http://tools.ietf.org/html/rfc2435.

[27] Mpeg-4, estándar (inglés). http://mpeg.chiariglione.org/standards/mpeg-4.

[28] OpenCV, página oficial (inglés). http://www.opencv.org.

[29] Pkg-config, repositorio de la aplicación (inglés). http://www.freedesktop.org/wiki/Software/pkg-config.

[30] Real Time Streaming Protocol (RTSP) (inglés). http://tools.ietf.org/html/rfc2326.

[31] TeX Live (inglés). http://www.tug.org/texlive/.

[32] Tiempo UNIX o POSIX, información relativa (inglés). http://unixtime.info/.

[33] Ubuntu, página oficial de la distribución de Linux Ubuntu (inglés). http://www.ubuntu.com/.

[34] Unified Modeling Language (UML), página oficial (inglés). http://www.uml.org/.

[35] Uniform Resource Locator (URL) (inglés). `http://tools.ietf.org/html/rfc3986`.

[36] VAPIX, API perteneciente al fabricante Axis Communications (inglés). `http://www.axis.com/vapix/`.

[37] Bjarne Stroustrup. *The C++ Programming Language*. Addison-Wesley, 2000.

[38] BOE. Ley Orgánica 15/1999, de 13 de diciembre, de Protección de Datos de Carácter Personal. 1999.

[39] BOE. Instrucción 1/2006, de 8 de noviembre. 2006.

[40] George T. Heineman, William T. Councill. *Component-based software engineering: putting the pieces together*. Addison-Wesley Professional, 2001.

[41] Ivar Jacobson, Grady Booch, James Rumbaugh. *The Unified Software Development Process*. Addison-Wesley Professional, 1999.