Índice general

Índice de figuras

Capítulo 1

Tecnologías usadas

Una vez introducidos en el proyecto y con los objetivos marcados, en este capítulo se hablará de las tecnologías usadas para el desarrollo de la aplicación más en profundidad. Las dos principales tecnologías en las que gira el proyecto son *Youtube live streaming API* y JdeRobot.

1.1. Streaming

Desde el punto de vista del usuario el uso del *streaming* es el siguiente. El usuario se conecta al servicio que le proporciona el contenido multimedia, dicho contenido se encuentra almacenado en servidores que contienen los archivos codificados en formatos como MP3,VP8, AVI... Una vez establecida la conexión comienza la transmisión que está basada en protocolos "ágiles" de transporte como RTSP, RTCP, UDP etc... La transmisión se hace en pequeñas partes de forma que se consigue una transmisión mas rápida al ser menos pesada. Por otro lado, para garantizar una reproducción continua se incluye un buffer en el cual van siendo almacenadas partes del contenido, de forma que el usuario no necesita descargar el contenido completo del archivo para empezar a reproducir, sino que solo necesita pequeñas partes de él.

En el proceso de *streaming* se deben tener en cuenta varios factores que pueden limitar la calidad del mismo. Desde el lado del consumidor el factor mas limitante es el ancho de banda, aunque con las nuevas velocidades de conexión casi cualquier compañía telefónica proporciona un ancho de banda suficiente.En el cuadro 1.1 se adjunta las recomendaciones de ancho de banda de Netflix en función de la calidad de vídeo.

CALIDAD	Mbps	GB POR HORA
STANDARD DEFINITION(SD)	3 MB	0,7 MB
HIGH DEFINITION(HD)	5 MB	3 MB
ULTRA HIGH DEFINITION(4K)	25 MB	7 GB

Cuadro 1.1: Ancho de banda recomendado en función de la calidad

Si nos situamos en el lado emisor se deben tener en cuenta otros factores:

- **Codec** , es un programa o dispositivo hardware capaz de codificar o decodificar una señal o flujo de datos digitales. El códec elegido afecta tanto a la calidad como a la velocidad de transmisión del archivo, ya que a mayor calidad de codificación mayor tasa de bits necesitamos. Uno de los mas usado es H264 para vídeo que está dejando paso a su sucesor H265, ambos permiten codificar vídeo de alta calidad.

- **BitRate**, el *bitrate* o tasa de bits representa la cantidad de bits que se envían por unidad de tiempo y es uno de los factores más importantes a la hora de producir una retransmisión de calidad aunque este factor está limitado por el ancho de banda de subida del que dispongamos. En este punto cabe destacar el uso del *multi-bitrate* que consiste en enviar distintas señales cada una con un *bitrate* diferente de forma que nos aseguramos que este contenido pueda ser consumido por todo tipo de conexiones.

- **Key Frame**, también conocido como *i-frame*, este fotograma representa una imagen completa y sirve de referencia a las demás imágenes en la que el codificador solo almacena las diferencias entre una y otra. Cuanto mayor sea el periodo de tiempo entre *KeyFrames*, menos datos se transmitirán pero peor será la calidad del vídeo.

- **FrameRate**, son las imágenes por segundo con las que se reproduce el vídeo. A mayor *framerate* mayor calidad y más pesado será el archivo.

1.1.1. Protocolos Streaming

Antes de hablar sobre los protocolo de *streaming*, se debe hacer una pequeña introducción sobre UDP y TCP, protocolos del nivel de transporte.UDP (User Datagram Protocol) se encarga de enviar datagramas a través de la red sin necesidad de establecer una conexión previa, tampoco posee información de flujo ni confirmación por lo que los paquetes pueden llegar desordenados o no llegar.

TCP (Transmission Control Protocol), su función al igual que UDP es la del transporte de datos. La diferencia entre ambos protocolos es que TCP además de requerir un establecimiento de conexión previo a al envió de datos, asegura que los paquetes llegarán ordenados y sin perdidas gracias a su mecanismo de asentimientos(ACK) y reenvío de paquetes. En su contra, el uso de este protocolo significa un mayor retardo en la comunicación frente a UDP.

Existen diversos protocolos para sostener la tecnología *streaming*, en esta sección vamos a centrarnos en dos tipos de protocolos. El primer grupo no se apoya en el protocolo HTTP para realizar el *streaming*, como son el protocolo RTP y RTMP. Por otro lado protocolos como DASH o HLS usan HTTP para *streaming*.

- RTP RTP son las siglas de Real-time Transport Protocol es un protocolo de nivel de sesión utilizado para la transmisión de información en tiempo real, como por ejemplo audio vídeo o datos. Junto a este protocolo se suele usar RTCP (RTP Control Protocol) es un protocolo de comunicación que proporciona

información de control que está asociado con un flujo de datos para una aplicación multimedia. Este protocolo no transporta ningún dato por si mismo se encarga de transmitir paquetes de control, datos de la conexión, bytes enviados, control de calidad ... Estos protocolos se encapsulan sobre UDP

RTSP es un protocolo de señalización (Real Time Streaming Protocol) establece y controla uno o muchos flujos sincronizados de datos, ya sean de audio o de vídeo. Es un protocolo no orientado a conexión, en lugar de esto el servidor mantiene una sesión asociada a un identificador, en la mayoría de los casos RTSP usa TCP para datos de control del reproductor y UDP para los datos de audio y vídeo. RTSP es similar a HTTP a excepción de que introduce nuevos métodos y necesita mantener el estado de la conexión. Los métodos mas importantes del protocolo son

- **Describe**, se solicita una descripción de un objeto multimedia contenido en el servidor. Con esta petición se comienza el diálogo.
- **Setup**, especifica como serán transportados los datos que suele incluir el puerto para recibir los datos RTP (audio y vídeo) y los datos de control RTCP.
- **Play**, esta petición provoca que el servidor comience a enviar el flujo de datos.
- **Pause**, detiene temporalmente el flujo de datos.
- **Teardown**, finaliza el envío de datos y libera los recursos usados.

- RTMP

  Protocolo de mensajería en tiempo real también conocido como *flash* y desarrollado por Adobe. Se trata de un protocolo basado en TCP que mantiene conexiones persistentes y comunicación en baja latencia. El flujo de información, en el cual se encuentran multiplexados datos multimedia y de conexión, es dividido en distintos fragmentos de forma que se entreguen flujos de información con la mayor cantidad de datos posible y sin problemas, este tamaño es negociado entre el cliente y el servidor. Por otro lado RTMP define varios canales para el intercambio de datos, estos canales pueden estar activo simultáneamente. Este encapsula por encima suya en MP3 o AAC el audio y en FLV el vídeo. Este protocolo presenta distintas variaciones puede usarse con conexiones TLS/SSL (RTMPS) junto con encriptación (RTMPE) , encapsulado dentro de HTTP (RTMPT) o sobre UDP (RTMFP).

- DASH

  Dynamic Adaptative Streamming Over HTTP también conocido como MPEG-DASH es un protocolo de *streaming* adaptativo cuyo objetivo es modular la tasa de bits en función del estado de la red. Para ello su idea principal es disponer del contenido en diferentes calidades y fragmentado de forma que cada segmento temporal puede ser enviado en distintas calidades. DASH usa HTTP como su protocolo de transporte lo que simplifica las conexiones a la hora de atravesar

nats o cortafuegos. Cabe destacar que este protocolo es un estándar abierto de W3C[1]

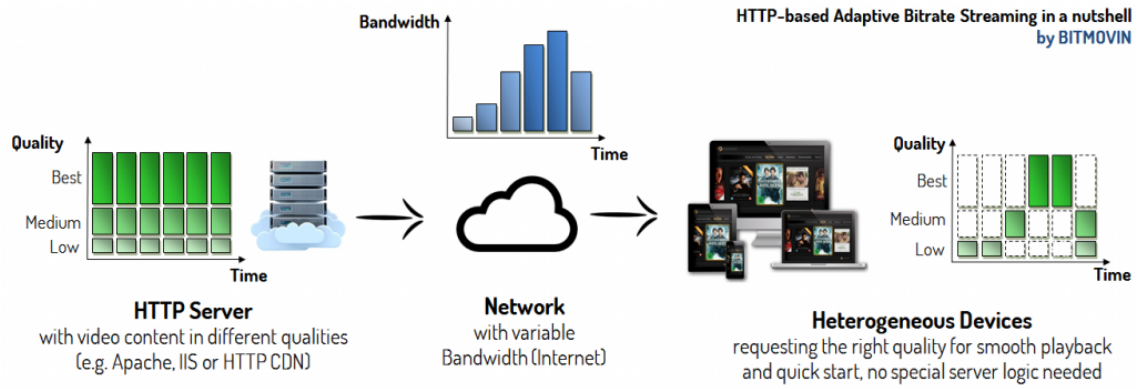

Figura 1.1: Arquitectura DASH

- HLS

  HTTP Live Streaming es un protocolo basado en HTTP implementado por Apple. Divide el flujo en pequeñas partes que son transmitidas, permitiendo al cliente a elegir el tipo de transmisión que más se adapte a su conexión. También implementa un mecanismo de codificación basado en AES.

  La arquitectura del protocolo se divide en un servidor que se encarga de codificar y encapsular la entrada de vídeo para ello utiliza H.264 como códec de vídeo y MP3, HE-AAC o AC-3 como códec de audio. Un distribuidor, que es un servidor web convencional, procesa las peticiones y devuelve los recursos pedidos y por último un cliente que recibe el flujo de vídeo.

[1]https://www.w3.org/2011/09/webtv/slides/W3C-Workshop.pd

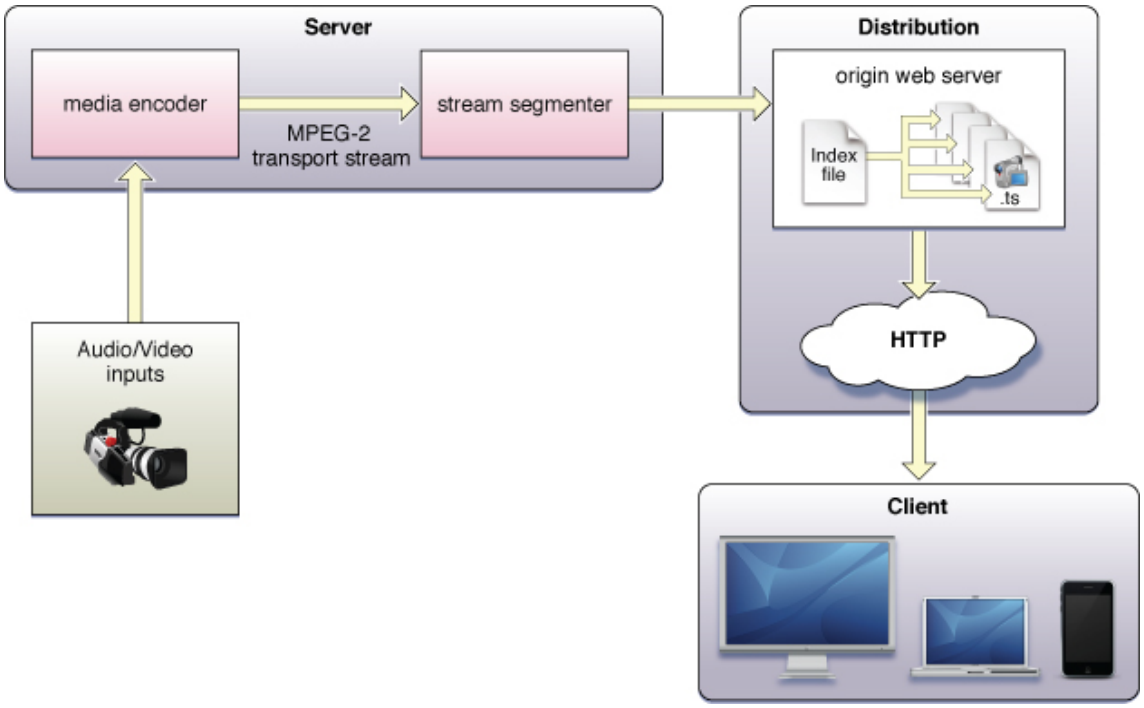

Figura 1.2: Arquitectura HLS

1.2. YouTube

Tras una breve introducción a YouTube en el primer capitulo, en este apartado revisaremos mas a fondo cómo funciona YouTube realmente y por otro lado el API de YouTube usada en el desarrollo de este proyecto.

Para comprender como funciona YouTube este apartado se dividirá en dos partes, en la primera se tratará el proceso de subida de vídeos a YouTube y la segunda parte se centrará en el proceso de reproducción.

1.2.1. Procesado del contenido subido

Los servidores de YouTube reciben vídeo en distintos formatos y calidades, no todos ellos son compatibles con los reproductores. Por YouTube convierte estos vídeos a un formato genérico capaz de ser reproducido. De forma que nunca vemos el vídeo originalmente subido ya que aunque este tenga una calidad excelente sería demasiado pesado lo cual ralentizaría su reproducción y por tanto el usuario no disfrutaría de una buena experiencia de visualización.

La primera parte de la transformación del vídeo es el procesado, en el que averigua la calidad y *framerate* del mismo. Con estos datos se genera un primer archivo comprimido mínimamente llamado *mezzanine* y que posteriormente será comprimido de nuevo.

El siguiente paso consiste en dividir este vídeo en fragmentos, suelen ser de unos cinco segundos, estos fragmentos son procesados y comprimidos en paralelo por dis-

tintas máquinas dando lugar a distintas versiones de cada uno en distintos formatos de compresión.

Por último, estos fragmentos se unen formando vídeos con distintas resoluciones, que son procesados por codecs con el fin de reducir aún más su tamaño.

A todo este proceso se le debe añadir un sotfware de análisis del contenido cuya función es buscar contenido protegido por derechos de autor o copyrigth, en caso de encontrarlo y el autor del vídeo no estar en posesión de dichos derechos ese contenido será eliminado por YouTube.

1.2.2. Reproducción

La segunda parte es la reproducción del contenido multimedia. En sus inicios YouTube tenia un único archivo de vídeo que era descargado y reproducido en la aplicación, este mecanismo generaba muchos retrasos y paros en la reproducción por lo que YouTube optó por dividir el vídeo en fragmentos. El tamaño de estos fragmentos se decide en función del estado de la red en lo que se llama *bitrate* adaptativo de forma que se minimicen las interrupciones.

Por otro lado la distribución de vídeo se hace a través del llamada red de distribución de YouTube que se encuentra asociado con los distintos proveedores de internet repartidos por distintas localizaciones. Estos proveedores almacenan cierto contenido multimedia, de forma que cuando hacemos la petición de un vídeo no nos comunicamos con los servidores de YouTube directamente sino con el proveedor más cercano. Si nuestro proveedor no posee el contenido requerido este hace una petición a servidores superiores de tal forma que se establece una red, donde el servidor superior es el centro de datos de Google.

1.2.3. YouTube Live Streaming

Esta API creada por Google, forma parte de YouTube data API y permite crear y manejar los eventos en vivo de YouTube. Desde ella puedes programar eventos y asociarlos a un flujo. Antes de continuar explicando el API vamos a introducir unos términos que facilitaran su compresión.

- *Broadcast*, representa una emisión de un flujo multimedia, en este caso dicho flujo será vídeo y audio. Youtube le da el nombre de eventos en vivo de forma que permite programarlos a una hora determinada. Dentro del API se encuentra asociado a un recurso llamado `liveBroadcast`.

- *Streams*, se trata del flujo multimedia, audio y vídeo. Cada *stream* se encuentra asociado a una emisión, dentro del API se puede acceder a él a través de `liveStream`.

YouTube proporciona varias librerías que se encuentran disponibles en diversos lenguajes, algunas de ellas en fase de pruebas aún como las de JavaScript. Las librerías estables se encuentran desarrolladas en java, Python y PHP. Para este proyecto por razones de compatibilidad se ha decido usar las librerías escritas en Python.

A continuación se explicaran unos puntos importantes para comprender mejor el uso del API, como son los mecanismos de seguridad, los eventos en vivo y el manejo de estos a través de las librerías.

Seguridad

Para usar las funcionalidades que nos proporciona el API en nuestra aplicación debemos registrar dicha aplicación a través de *Google Developers Console*. Allí se conseguirán unas credenciales que darán acceso a su uso. Esto es necesario ya que acceder a YouTube, en este caso, significa acceder a un sitio privado con datos de usuario sensibles por lo que se necesitan mecanismos que los protejan.

Para este fin Google ha elegido el protocolo de seguridad Oauth 2.0 (Open Authorization). Dicho protocolo permite flujos simples de autorización para sitios web o aplicaciones informáticas. Además permite a terceros (clientes) acceder a contenidos propiedad de un usuario (alojados en aplicaciones de confianza, servidor de recursos) sin que éstos tengan que manejar ni conocer las credenciales del usuario. Es decir, aplicaciones de terceros, en este caso nuestro proyecto, pueden acceder a contenidos propiedad del usuario, nuestra cuenta de YouTube, pero estas aplicaciones no conocen las credenciales de autenticación.

La primera vez que accedes a tu aplicación a través del API serás redirigido al servidor de autenticación de Google donde debes dar tu autorización para que la aplicación puede acceder a tus recursos de usuario, una vez aceptado se genera un *token* usado posteriormente por Oauth 2.0.

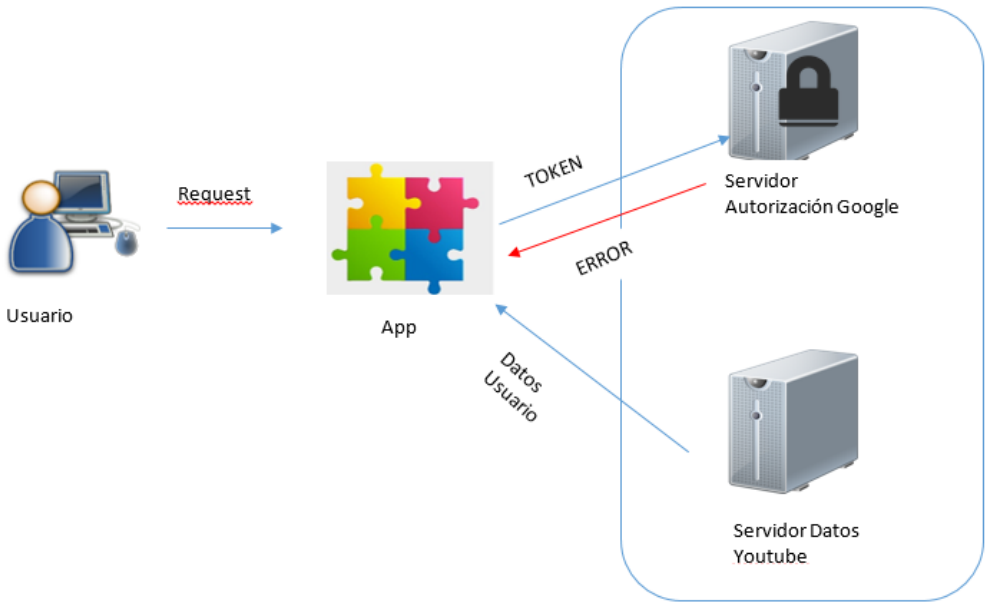

Figura 1.3: Protocolo Oauth YouTube

Recursos YouTube API: liveBroadcast y liveStream

La API presenta distintos recursos. A continuación presentaremos los recursos usado para este proyecto y cómo a través de su manipulación podemos manejar eventos en vivo desde una aplicación.

El primero, `liveBroadcast`, como su propio nombre indica, se encarga de manejar la información relacionada con la emisión del evento. Este recurso tiene asociadas ciertas propiedades como la hora del evento o la privacidad. Estas propiedades pueden ser definidas mediante un archivo en formato JSON, así a través de este archivo se puede configurar la emisión.

```
{
  "kind": "youtube#liveBroadcast",
  "etag": etag,
  "id": string,
  "snippet": {
    "publishedAt": datetime,
    "channelId": string,
    "title": string,
    "description": string,
    "thumbnails": {
      (key): {
        "url": string,
        "width": unsigned integer,
        "height": unsigned integer
      }
    },
    "scheduledStartTime": datetime,
    "scheduledEndTime": datetime,
    "actualStartTime": datetime,
    "actualEndTime": datetime,
    "isDefaultBroadcast": boolean,
    "liveChatId": string
  },
  "status": {
    "lifeCycleStatus": string,
    "privacyStatus": string,
    "recordingStatus": string,
  },
  "contentDetails": {
    "boundStreamId": string,
    "boundStreamLastUpdateTimeMs": datetime,
    "monitorStream": {
      "enableMonitorStream": boolean,
      "broadcastStreamDelayMs": unsigned integer,
      "embedHtml": string
    },
    "enableEmbed": boolean,
    "enableDvr": boolean,
    "enableContentEncryption": boolean,
    "startWithSlate": boolean,
    "recordFromStart": boolean,
    "enableClosedCaptions": boolean,
```

```
    "closedCaptionsType": string,
    "projection": string,
    "enableLowLatency": boolean
  },
  "statistics": {
    "totalChatCount": unsigned long
  }
}
```

Para manipular estos datos el recurso nos proporciona varios métodos, aquí solo se explicarán algunos.

- **Insert**, este método crea un evento. Para ello debe recibir mínimo dos parámetros de entrada, el primero hace referencia a las partes del JSON que serán definidas(snipet, status, contentDetails. . . .), mientras que el segundo argumento que recibe es el propio JSON con los datos. Se deben proporcionar obligatoriamente el título, la hora de inicio y el estado de privacidad del vídeo. Los demás datos si no están definidos tomarán un valor por defecto. Como respuesta a este método obtenemos el mismo JSON pero esta vez todos los campos con su valor asociado, ya sea el que le hemos asignado o uno por defecto. Dentro de las propiedades obtendremos el ID del recurso que será necesario en el método BIND para poder asociarlo a un flujo.

- **List**, este método retorna una lista con todos los datos de los *broadcast* pedidos en función de unos parámetros de entrada. Dentro de los parámetros obligatoriamente debemos especificar las propiedades del recurso que queremos recuperar. Adicionalmente podemos aplicar ciertos filtros a los resultados.

  - *broadcastStatus*, devuelve únicamente las emisiones que se encuentren en el estado determinado, las opciones son activo, todos, completado o sin comenzar
  - *id*, es el filtro más específico devuelve únicamente el recurso asociado a un identificador
  - *maxResutls*, limita los resultados obtenidos, por defecto tiene un valor de cinco pero puede tomar valores entre cero y cincuenta.
  - *broadcastType*, con este filtro estipulamos que queremos obtener eventos de un tipo determinado siendo las opciones evento, una retransmisión programada a una hora determinada, persistente, un evento el cual se encuentra continuamente activo, u ambos.

- **Bind**, este método puede realizar dos acciones en función de los parámetros que reciba. Obligatoriamente debe recibir las partes del recurso, y un *id* perteneciente a un *broadcast*. Opcionalmente puede recibir un *id* que representa un recurso `livestream`, si recibe este parámetro el método enlazará ambos recursos quedando asignado a la emisión un flujo de vídeo, si por el contrario este parámetro no es proporcionado el método *bind* desenlazará de la emisión el flujo de vídeo si esta la tuviera.

- **Transition**, una vez creados y enlazados ambos recursos este método nos da la posibilidad de cambiar el estado de la emisión. Es decir podemos hacer pública la emisión, pasar a estado de test o dar por finalizada la emisión. Para ellos deberemos proporcionarle el estado el cual queremos dar a nuestra emisión, el *id* de dicha emisión y las partes del recurso que queremos obtener en la respuesta.

El segundo recurso que nos encontramos es `liveStream` que nos permite configurar las propiedades de la ingestión del vídeo.Los métodos de este recurso son similares a los métodos del `liveBroadcast` con la diferencia de que son asociados a un *stream* y sus propiedades se definen mediante un JSON que se puede ver a continuación.

```
{
  "kind": "youtube#liveStream",
  "etag": etag,
  "id": string,
  "snippet": {
    "publishedAt": datetime,
    "channelId": string,
    "title": string,
    "description": string,
    "isDefaultStream": boolean
  },
  "cdn": {
    "format": string,
    "ingestionType": string,
    "ingestionInfo": {
      "streamName": string,
      "ingestionAddress": string,
      "backupIngestionAddress": string
    },
    "resolution": string,
    "frameRate": string
    "status": {
    "streamStatus": string,
    "healthStatus": {
      "status": string,
      "lastUpdateTimeSeconds": unsigned long,
      "configurationIssues": [{
          "type": string,
          "severity": string,
          "reason": string,
          "description": string
  }]}
    },
    "contentDetails": {
        "closedCaptionsIngestionUrl": string,
        "isReusable": boolean
    }
}
```

El método `list` es prácticamente igual al explicado para `liveBroadcast` a diferencia que el resultado que obtenemos del mismo es una lista de *streams*. Respecto al

método *insert* su función es la de crear el recurso siendo las propiedades obligatorias a definir el título, el formato y el tipo de ingestión. Esta última propiedad, el tipo de ingestión, establece el protocolo por el cual se transmite el flujo de vídeo a YouTube. En este campo YouTube nos proporciona dos protocolos distintos DASH o RTMP.

1.3. FFmpeg

FFmpeg [2] es una colección de librerías de software libre capaz de decodificar, codificar, transcodificar, multiplexar, demultiplexar, filtrar y reproducir gran cantidad de archivos en múltiples formatos, también se encuentra disponible para distintos sistemas operativos como Linux, Mac OS X, Windows ... para la mayoría de sus distribuciones.

FFmpeg presenta cuatro herramientas distintas:

- **ffmpeg**, herramienta que proporciona un rápida conversión de archivos de audio y vídeo a través de la línea de comandos.
- **ffserver**, formado por un servidor de *streaming* para audio y vídeo. Es capaz de soportar varios canales en vivo y *streaming* de archivos.
- **ffplay**, es un reproductor multimedia simple basado en SDL (Simplec DirectMedia Layer) y las bibliotecas de ffmpeg.
- **ffprob**, se encarga de reunir información de streams multimedia como formatos, bitrates, framerates...

En este proyecto únicamente se hará uso de ffmpeg como línea de comando ya que mediante esta herramienta podremos enviar flujo de vídeo al servidor de YouTube.

Otro punto importante a la hora de hablar de FFmpeg son las librerías:

- **libavutil**, constituye una biblioteca de apoyo de forma que se simplifica la programación incluyendo estructuras de datos, rutinas matemáticas, utilidades capa multimedia y otras muchas.
- **livavcodec**, esta librería está formada por codificadores y decodificadores de audio y vídeo.
- **libavformat**, es la parte encargada de multiplexación y demultiplexacion para diferentes formatos multimedia.
- **libavdevice**, esta librería contiene herramientas de entrada y salida para grabar y renderizar el contenido multimedia generado por entornos como Video4Linux, Vfm o ALSA.
- **libavfilters**, proporciona filtros para contenido multimedia, filtros paso bajo, paso alto, compresores , bicuadrados ...

[2] https://ffmpeg.org/

- **livwscale**, esta librería realiza operaciones altamente optimizadas de escalado de imagen y espacio de color.
- **libswresample**, es capaz de hacer muestreo y conversiones de formato.

Como podemos ver FFmpeg es una herramienta muy potente y que nos proporciona diversas opciones.

1.3.1. FFmpeg streaming

FFmpeg proporciona dos caminos para realizar *streaming* el primero consiste en enviar directamente el flujo de vídeo a un servidor, YouTube en nuestro caso, y este retransmitirlo nuevamente. La otra alternativa consiste en retransmitir directamente a un usuario final, incluso a través de la creación de múltiples salidas retransmitir hacía más de un usuario.

La herramienta ffmpeg a través de la línea de comandos nos permite enviar un flujo de vídeo al servidor de YouTube. Para esto hemos elegido el protocolo de transporte RTMP, dicho protocolo es mencionado en la introducción, este protocolo es capaz de intercambiar un flujo multimedia entre un reproductor flash y un servidor. El formato elegido para el vídeo es FLV (flash video player) ya que es el códec de vídeo usado por YouTube.

El sistema operativo Linux, nos ofrece algunas herramientas que complementan a ffmpeg como puede ser *video4linux* , que consiste en un API de captura de vídeo para Linux capaz de capturar la imagen de una webcam. Por otro lado para capturas de audio, ALSA (Advanced Linux Sound Architecture) es una buena alternativa, constituye un controlador de sonido del núcleo de Linux.

Como se podrá ver más adelante combinando todas estas herramientas que pone a nuestra disposición ffmpeg conseguiremos retransmitir un flujo de vídeo hacia YouTube.

1.4. Open Broadcaster Software

Esta aplicación conocida por sus siglas OBS [3], constituye un sofware libre y de código abierto para la grabación y retransmisión (streaming) de vídeo. Para este proyecto se ha usado la versión 0.15.1 de OBS.

OBS comenzó como un pequeño proyecto creado por Hugh "Jim"Bailey, pero creció rápidamente con la ayuda de muchos colaboradores que trabajan para mejorar la aplicación. En 2014,comenzó a desarrollarse una nueva versión conocida como OBS Multiplatform (más tarde renombrada OBS Studio) para soporte multiplataforma, siendo un programa más completo y con una API más potente.OBS Studio es un trabajo en progreso ya que, a febrero de 2016, no ha alcanzado la paridad de características con el original de la OBS, es por eso que el original aun está disponible en el sitio.

[3] https://obsproject.com/

Este proyecto usa como lenguaje de programación C y C++, permite capturar fuentes de vídeo en tiempo real, composición de escenas, codificación, grabación y retransmisión. Para la retransmisión de contenido en vivo usa RTMP como protocolo de transporte , de forma que es compatible con YouTube o Twitch entre otras plataformas.

Además la comunidad OBS ha desarrollado varios plugins que extienden sus funcionalidades como OBS remote que permite acceder remotamente a OBS a través de internet o CLR Browser Source que permite usar como fuente de vídeo un recurso web.

Este software permite realizar las mismas operaciones que ffmpeg pero de una forma mas sencilla debido a su interfaz gráfico, por lo que para usuarios menos expertos es una mejor alternativa que ffmpeg.

Figura 1.4: Interfaz OBS

1.5. JdeRobot

JdeRobot [4] es un entorno de desarrollo para aplicaciones robóticas y de visión por computador. Este entorno, desarrollado por el grupo de robótica de la universidad Rey Juan Carlos, simplifica el acceso a los sensores, actuadores o robots como el drone. JdeRobot ha sido desarrollado en su gran mayoría usando como lenguaje de programación C++, basándose en un entorno de componentes distribuidos que pueden ser escritos en otros lenguajes como Java o Python. Finalmente estos componentes se comunican usando ICE.

ICE hace referencia a las siglas de Internet Communications Engine y consiste en un middleware orientado a objetos desarrollado por la empresa ZeroC [5] , que proporciona ayuda a la hora de desarrollar aplicaciones distribuidas ya que se encarga de todas las

[4]http://jderobot.org
[5]https://zeroc.com

interacciones con los interfaces de red tales como abrir conexiones de red, serializar y deserializar datos o reintentos de conexiones fallidas. Gracias a esto se puede crear conexiones entre distintas máquinas con distintos sistemas operativos o lenguajes de programación.

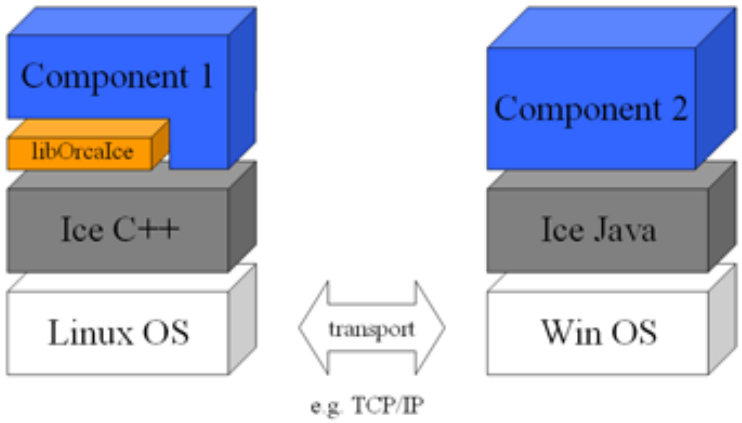

Figura 1.5: Estructura ICE

JdeRobot también usa una distribución de ICE, llamada ICEJs que nos da la opción de conectar navegadores usando JavaScript y los protocolos de ICE a través de *websockets*.

Recientemente JdeRobot ha lanzado su última versión estable JdeRobot 5.4.1. Esta versión aporta varios cambios que facilitan aún más el uso de sus herramientas. A continuación se comentan los cambios más significativos:

- Cambio de SO, la nueva versión de JdeRobot es soportada únicamente por Ubuntu 16.04.
- Cambio en la versión de Gazebo, la versión antigua trabajaba con la versión 5 de Gazebo, con los nuevos cambios se ha pasado a usar la versión 7.
- Actualización de la versión de ICE, este punto es importante ya que anteriormente JdeRobot trabajaba con la versión 3.5 de ICE en la cual no venía incorporado el plugin ICEJS lo que implicaba una mayor dificultad a la hora de instalar las librerías así como problemas de compatibilidad. En la nueva versión se ha pasado a usar la versión 3.6 de ICE donde si está incorporado el plugin citado y que resulta de gran utilidad para este trabajo fin de grado.

JdeRobot ha desarrollado distintos componentes y plugins, en este proyecto se usarán dos de ellos principalmente CameraServer y ArDroneServer.

1.5.1. ArDroneServer

ArDroneServer es un componente cuya función es la comunicación a bajo nivel con un drone real. Este componente a través de seis interfaces ICE es capaz de recoger

datos y valores provenientes de los sensores y enviar ordenes a los motores y actuadores del drone en función de las instrucciones de movimiento que le lleguen. Las interfaces son las siguientes.

- `cmd_vel`, interfaz para los comando de velocidad
- `navdata`, datos sensoriales como altitud o velocidades
- `ardrone`$_e xtra, interfaz para funciones básicas y extra que presenta el dron e como el aterrizaje, desp$
- `remoteConfig`, interfaz estándar para la transmisión de ficheros XML en JdeRobot.
- `Pose3D`, esta interfaz recoge datos de posicionamiento en forma vectorial(x,y,z)

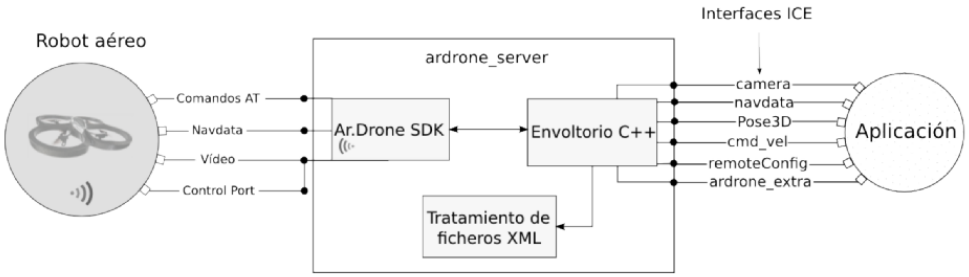

Figura 1.6: Arquitectura ArDroneServer

1.5.2. CameraServer

CameraServer es un componente desarrollado por JdeRobot capaz de servir N cámaras tanto reales como simuladas. Para el manejo de los vídeos, internamente usa *gstreamer*, entorno multimedia libre multiplataforma escrito en el lenguaje de programación C, permite crear aplicaciones audiovisuales, como de vídeo, sonido, codificación etc.

CameraServer permite captar el vídeo de diferentes fuentes como puede ser video4linux, un archivo o un recurso web.

CameraServer proporciona una interfaz, Camera Interface, esta interfaz aporta los métodos de configuración de la cámara así como los métodos de inicio y final de la toma de imágenes. Dicho interfaz extiende otro llamado ImageProvider el cual aporta métodos de configuración de la imagen, como el formato, y métodos que recuperan imágenes de forma cruda, sin comprimir, fotograma a fotograma y son entregados al interfaz Camera.

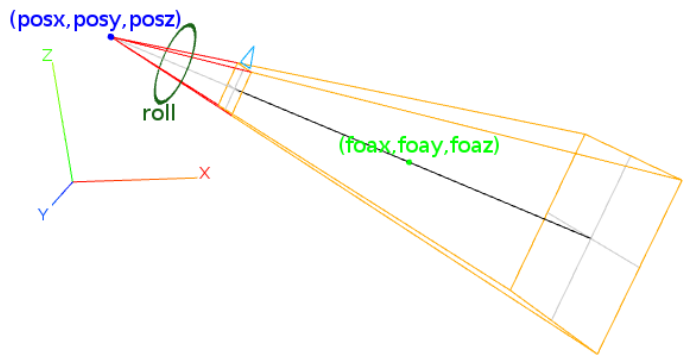

Figura 1.7: Modelo CameraServer

1.5.3. Plugins de Gazebo que incluyen Camera

JdeRobot ha desarrollado varios plugins usando Gazebo. Estos plugins facilitan la programación a más bajo nivel, como son las conexiones de forma que podemos manejar el drone en un mundo creado por Gazebo y depurar nuestro código.

Algunos de los plugins nos permite manejar drones con cámara integrada como es el caso de*Quadrotor2 Gazebo* [6], el cual simula un quadrotor con una cámara incorporada de la cual se pueden recoger las imágenes fotograma a fotograma a través del interfaz Camera.

Otro ejemplo es kobuki_driver, donde se simula un Turtlebot, un pequeño robot con ruedas. El plugin de JdeRobot permite conducirlo a través de un mundo simulado a la vez que se accede a las imágenes captadas por una cámara incorporada a él.

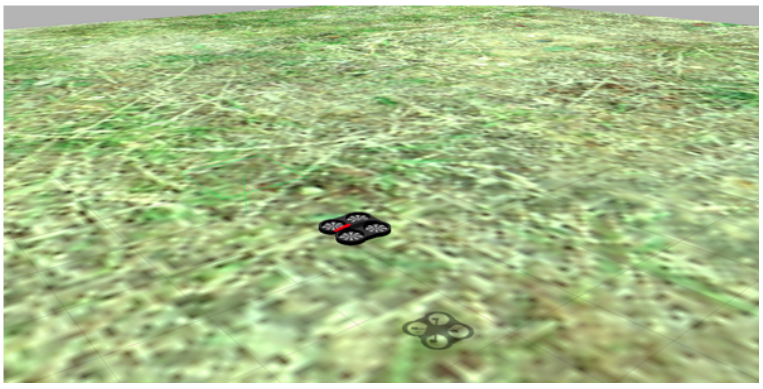

Figura 1.8: Escenario Quadrotor2 Gazebo

1.6. Gazebo

Gazebo es un simulador 3D, que permite crear escenarios en tres dimensiones en tu ordenador con robots, obstáculos y otros muchos objetos. Gazebo fue diseñado para

[6]http://jderobot.org/index.php/DriversQuadrotor2$_{G}azebo$

evaluar algoritmos y comportamientos de robots en un escenario simulado pero muy cercano a la realidad sin exponer a peligros a los robots. Para muchas aplicaciones es esencial testear las aplicaciones de forma que se pueden evitar fallos e batería, de manejo, localización o manejo entre otros. Es un proyecto de código abierto y actualmente cuenta con una gran comunidad de desarrolladores.

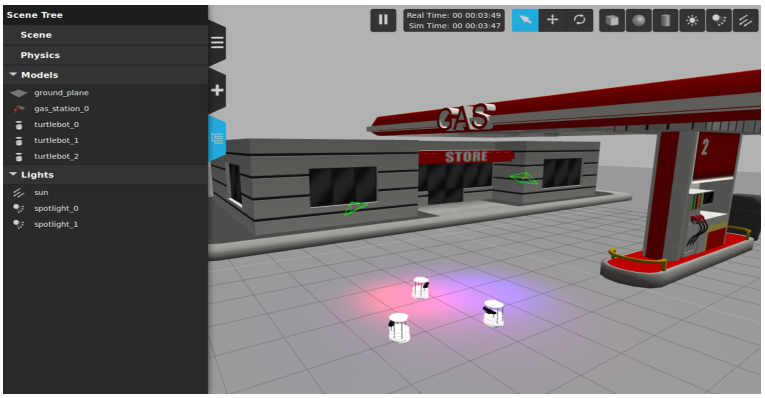

Figura 1.9: Escenario creado con gazebo

Gazebo ha sido útil para este proyecto a la hora de simular el comportamiento de un drone con una cámara a bordo que nos proporciona las imágenes necesarias para poder enviar el flujo multimedia a YouTube.