3. Tecnología software

Una vez introducidos en el proyecto y con los objetivos marcados se hablara 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 streamming API* y DroneWebRTC apoyada en Gazebo, webRTC o JdeRobot entre otras.

3.1 Youtube live streamming

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 asociarlo a un *stream.* 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 video 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 video. Cada *stream* se encuentra asociado a una emisión, dentro del API se pude acceder a él a través de liveStream.

YouTube proporciona varias librerías para poder interactuar dichas librerías se encuentran escritos en diversos lenguajes algunas de ellas en fase de pruebas aun 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.

3.1.1 Seguridad

Para poder usar las funcionalidades que nos proporciona el API en nuestra aplicación debemos registrar dicha aplicación a través de *Google Developers Console*. Donde conseguiremos unas credenciales que nos 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 y de esta forma se evita que cualquiera pueda acceder a tus datos.

El protocolo de seguridad usado es el Oauth 2.0 (*Open Authorization), es un protocolo* es un protocolo que permite flujos simples de autorización para sitios web o aplicaciones informáticas, 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. A continuación se presenta un esquema del protocolo.

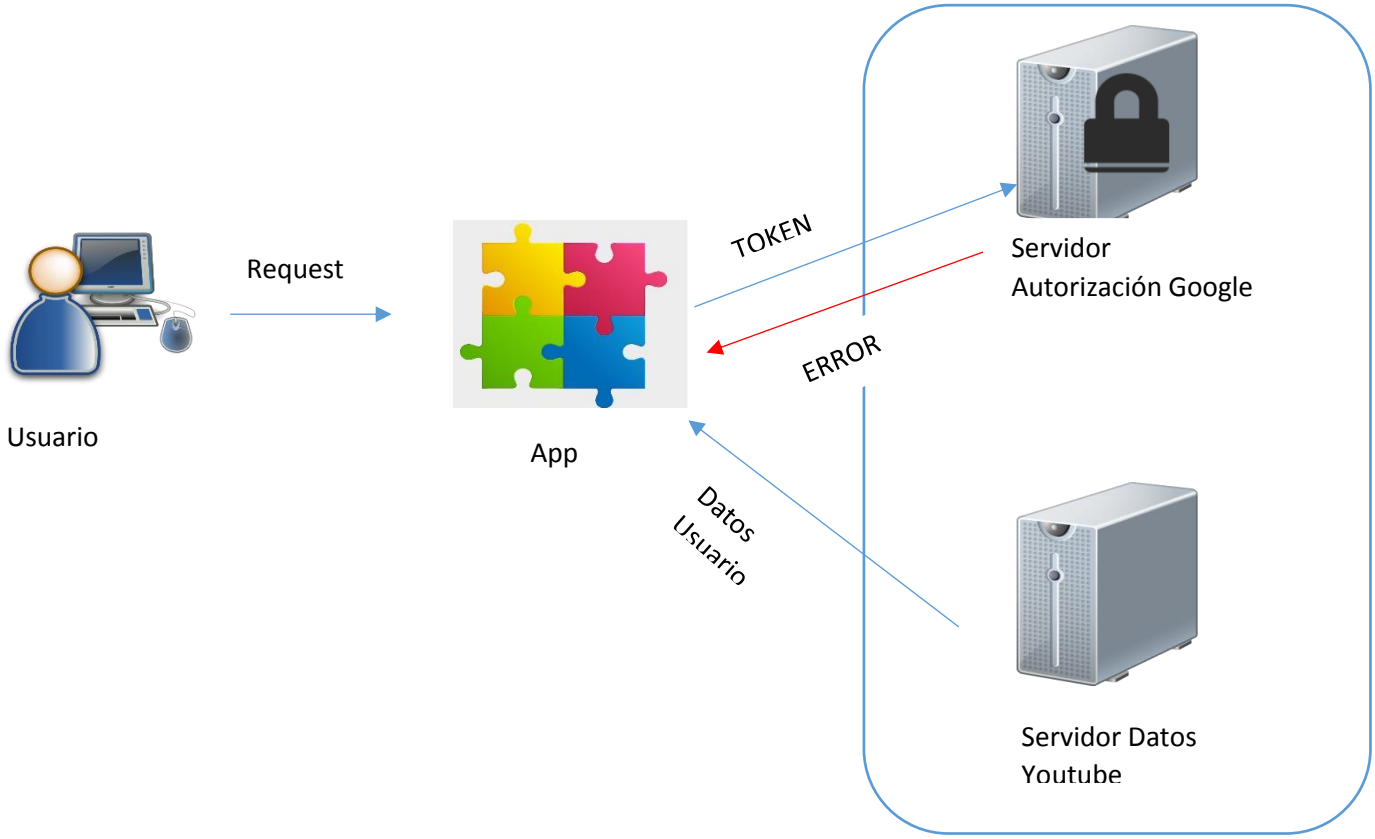

Figura 1.1 Esquema protocolo Oauth

3.1.2 LiveBroadcast y LiveStream

Ambos son recursos que nos proporcionan las librerías de la API para manejar los eventos en vivo.

El primero *liveBroadcast* como su propio nombre indica se encarga de manejar la información del evento. Esta información se encuentra almacenada en un JSON que podemos observar a continuación.

```
{
  "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
  }
}
```

Figura 3.2 JSON que define propiedades de liveBroadcast

Para manipular estos datos el recurso nos proporciona varios métodos, aquí solo se explicaran 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 se puede observar en la figura 3.2 (snnipet, 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 video. Como respuesta a este método obtenemos el mismo JSON pero con todos los campos rellenos con los datos que hemos proporcionado o unos por defecto si estos no han sido proporcionados. Una propiedad importante es el ID del recurso ya que será necesario en el método *BIND* para poder asociarlo a un stream.

- ***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 partes del recurso, figura 3.2, 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 porgramada 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 enlazara ambos recursos quedando asignado a la emisión un flujo de video, si por el contrario este parámetro no es proporcionado el método *bind* desenlazara de la emisión el flujo de video 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 la ingestión del video manipulando un archivo JSON incluido 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
  }
}
```

Figura 3.3 JSON que define propiedades de liveStream

Los métodos asociados a este recurso son similares a los métodos del *liveBroadcast* con la diferencia de que son asociados a un *stream.* Dentro de este recurso cabe destacar la propiedad *ingestionType*, contenida en el JSON adjuntado anteriormente. Esta propiedad establece el protocolo por el cual se transmite el flujo de video a YouTube. En este campo YouTube nos proporciona dos protocolos distintos DASH o RTMP, en este proyecto el protocolo usado será RTMP debido a una mayor compatibilidad con *ffmpeg*, herramienta que codifica y transmite el flujo de video al servidor de YouTube, dicha herramienta es explicada a continuación.

3.2 FFMPEG

FFmpeg 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, que son enumeradas a continuación.

- ***ffmpeg,*** herramienta que proporciona un rápida conversión de archivos de audio y video a través de la línea de comandos
- ***ffserver,*** formado por un servidor de streamming para audio y video, es capaz de soportar varios canales en vivo y streamming 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 video al servidor de YouTube como se verá más adelante.

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 video.
- ***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 *frameworks* como *Video4Linux, Vfm o ALSA.*
- ***libavfilters,*** proporciona filtros para contenido multimedia, filtros paso bajo, paso alto, compresores , bicuadrados …
- ***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.

3.2.1 FFmpeg streaming

FFmpeg proporciona dos caminos para realizar streaming el primero y el usado en este proyecto consiste en enviar directamente el flujo de video 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 podría ser posible retransmitir hacía más de un usuario.

La herramienta *ffmpeg* a través de la línea de comandos nos permite enviar un flujo de video 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 video es FLV (flash video player) ya que es el códec de video usado por YouTube.

Para la captura del flujo multimedia se ha optado por usar *video4linux2* para el video. Es un API de captura de video para Linux y es capaz de capturar la imagen de una webcam. Por otro lado para capturar el audio se ha elegido ALSA (Advanced Linux Sound Architecture) que es un controlador de sonido del núcleo de Linux.

Figura 3.4 Esquema retransmisión Streaming FFmpeg

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

3.3 Drone WebRTC

Como se comentó en la introducción este es un proyecto desarrollado anteriormente por otro alumno de la URJC. No es una tecnología como tal pero servirá de marco para introducir otras tecnologías usadas. Primero se hará una breve descripción de dicho proyecto tras el cual se explicarán las tecnologías.

Drone WebRTC consiste en una aplicación web que nos permite tele operar un dron. Para conseguirlo se ha hecho uso de webRTC de forma que a través de la interfaz RTCDataChannel se transmiten instrucciones de movimiento o despegue, dadas en un cliente remoto, estas instrucciones son recogidas en un par local que se comunica con el dron directamente a través de websockets y usando las herramientas proporcionadas por JdeRobot apoyadas en Gazebo e ICE.

El proyecto original únicamente se centraba en retransmitir video y utilizaba una versión antigua de JdeRobot, ICE y Gazebo. En este trabajo aprovecharemos el desarrollo llevado a cabo para actualizarlo a nuevas versiones además de añadirle audio.

INSERTAR ARQUITECTURA

3.1.1 WebRTC

RTC son las siglas de *Real Time Communication* y como su propio nombre indica constituye un proyecto de código abierto que permite comunicaciones en tiempo real entre dos navegadores sin necesidad de ningún plugin adicional ni servidores intermedios excepto para la señalización. Todas estas operaciones se realizan gracias a tres API implementadas en JavaScript.

- **MediaStream,** este API es la encargar de acceder a la cámara y al micrófono capturando así el flujo multimedia. Una vez es capturado este flujo es procesado y se llevan a cabo mejoras de calidad ajuste de bitrate, sincronizaciones… todo ello de forma automática.
  Para conseguir este objetivo se hace uso del objeto *getUserMedia,* que implementa una función que toma el mismo nombre, y recibe tres parámetros dos *callback,* uno para el caso de fallo y otro para el éxito y un objeto *constrains* que representa las restricciones de los flujos de audio y video.

  IMAGEN DE CONSTRAINS

- **RTCPeerConnection,** es un API que se encargar de crear y manejar una conexión entre pares estable y eficiente. Algunas de sus funciones son:
  - Ocultamiento de paquetes recibidos
  - Cancelación de eco.
  - Adaptación del ancho de banda
  - Buffer dinámico
  - Reducción o eliminación de ruido

  Para poder hacer posible dicha conexión primeramente necesitamos poner en contacto ambas partes a través de un proceso de señalización explicado más adelante. Una vez hecho esto el protocolo de transporte usado por *RTCPeerConnection* es UDP, ya que aunque no garantiza la llegada de paquetes, al hablar de aplicaciones en tiempo real se priman la baja latencia y vivacidad por encima de la fiabilidad. Pero UDP no es suficiente para garantizar una buena conexión necesita apoyarse en servidores STUN, TURN para poder atravesar NAT's.

- **RTCDataChannel,** esta tercera API constituye un canal una vez establecida la conexión entre ambos pares, por el cual podemos transmitir todo tipo de datos. Los protocolos usados para ello son TCP, UDP y SCTP de forma que a partir de una variable de configuración podemos elegir el protocolo usado en función de nuestras necesidades. Todos estos datos se encuentran protegidos por protocolos de seguridad como DTLS y RTSP de forma que se encriptan todos los datos compartidos, esto se aplica a todos los componentes de webRTC.

Una vez explicadas las tres principales API´s de webRTC se incluye una imagen de la arquitectura que sigue. A continuación también se hablara del proceso de establecimiento de la conexión y los protocolos que ayudan a ello.

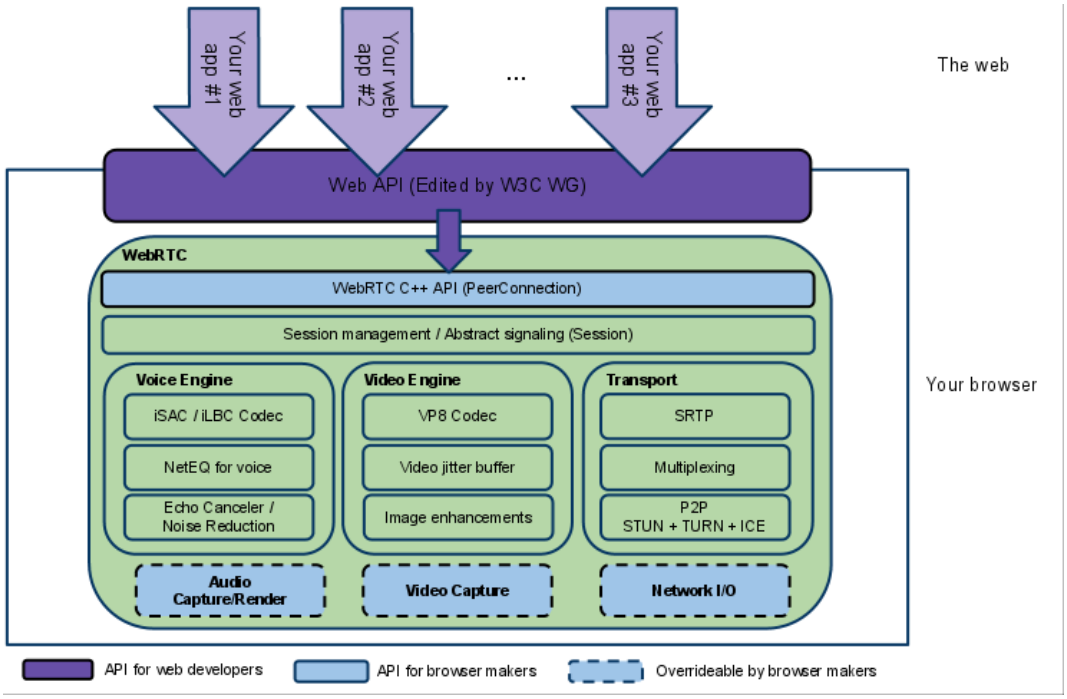

Figura 3.5 Arquitectura Aplicación webRTC

3.1.2 Establecimiento de conexión

Como se ha comentado anteriormente la finalidad de webRTC es la comunicación entre dos navegadores. Para que esta comunicación sea posible primero se debe establecer una conexión entre ambas partes de forma que haya un intercambio de datos y metadatos entre ambos. Es un proceso simple de oferta/respuesta entre un llamador (*caller)* y un llamado (*callee)*

Este intercambio no está definido en la API y da flexibilidad a la hora de hacerlo, la única restricción es seguir la arquitectura JSEP (*JavaScritp Session Estabkishment Protocol),* que hace uso de ICE y SDP.

- **SDP,** *Session Description Protocol* es un protocolo que describe los parámetros de inicialización de flujos multimedia, cubre aspectos como el anuncio de sesión, invitación a sesión y negociación de parámetros.
- **ICE,** *Interactive Connection Establishment* define un protocolo de actuación que proporciona el camino para que dos pares puedan iniciar una conversación a través de NAT's.

El proceso de señalización es un proceso complicado, el primer paso consiste en reunir los candidatos a través del protocolo ICE. Los llamados candidatos son posibles rutas de comunicación entre ambos pares. La problemática de este proceso es que la mayoría de equipos se encuentran en una red privada o detrás de cortafuegos por ello el protocolo ICE nos ayuda a descubrir la topología de la red y facilita la conexión, pero a menudo con esto no es suficiente debido a que descubrir ip´s y puertos en una red local es sencillo pero una vez fuera de la red es más complicado por ello el protocolo ICE a menudo se apoya en servidores STUN y TURN. El primero de ellos se encarga de proporcionar ip externas a la red mientras que los servidores TURN se encargan de redirigir el tráfico si la conexión directa falla. Primeramente el protocolo ICE intenta usar los candidatos encontrados por él mismo, si esto falla se recurre a un servidor STUN y si no se puede establecer la conexión el flujo se transmite a través del servidor TURN.

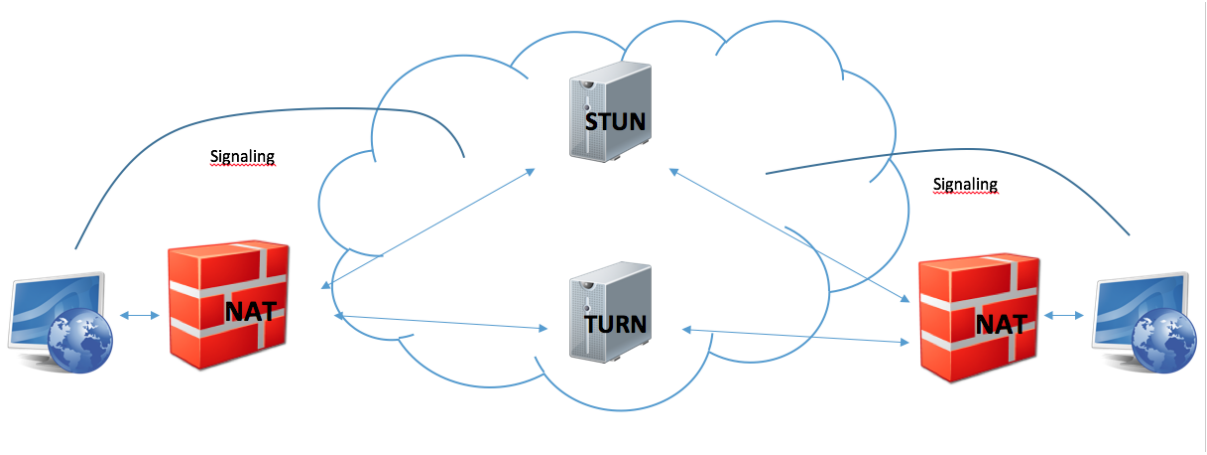

Figura 3.6 Comunicación a través de NAT's

Una vez obtenidos los candidatos se pasa al establecimiento de la sesión a través de una oferta SDP. Cabe destacar que el proceso de encontrar candidatos a veces puede ser un proceso lento de forma por lo que webRTC soporta *ICE Candidate Trickling,* que da la posibilidad al llamado a actuar en la llamada y comenzar a establecer una conexión sin esperar a que lleguen todos los candidatos

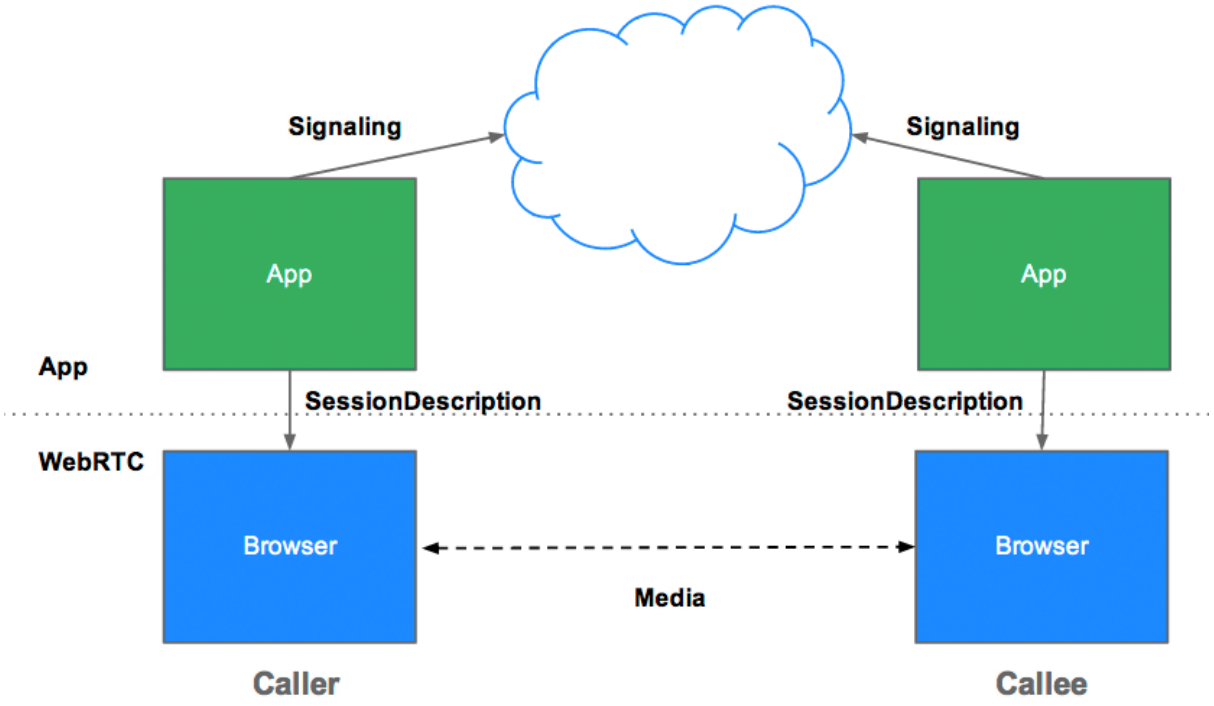

Figura 2.7 Señalización webRTC

3.3 JdeRobot

JdeRobot[1] es un entorno de desarrollo para aplicaciones robóticas y 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 unidades hardware del dron. 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[2], que proporciona ayuda a la hora de desarrollar aplicaciones distribuidas ya que se encarga de todas las 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 maquinas con distintos sistemas operativos o lenguajes de programación.

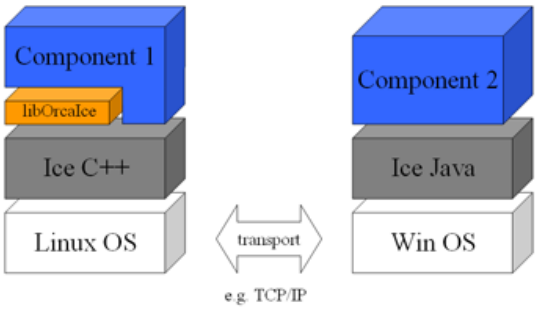

Figura 3.8 Estructura ICE

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.*

JdeRobot ha desarrollado distintos componentes y plugins, en este proyecto se usarán dos de ellos principalmente *ArDroneServer* y un plugin desarrollado para *Gazebo*.

ArDroneServer, es un componente capaz de comunicarse con un ArDrone real. ArDrone será el modelo de dron usado para este proyecto, es un cuadricoptero de la marca Parrot. Este componente facilita el acceso y la conexión con los sensores y mecánica del dron, de forma que se definen seis interfaces ICE configurables para manejar el dron.

A continuación para poder comprender completamente las tecnologías de este proyecto vamos a hablar sobre Gazebo y su relación con JdeRobot y la última versión de JdeRobot.

[1] http://jderobot.org
[2] https://zeroc.com

3.3.1 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 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.

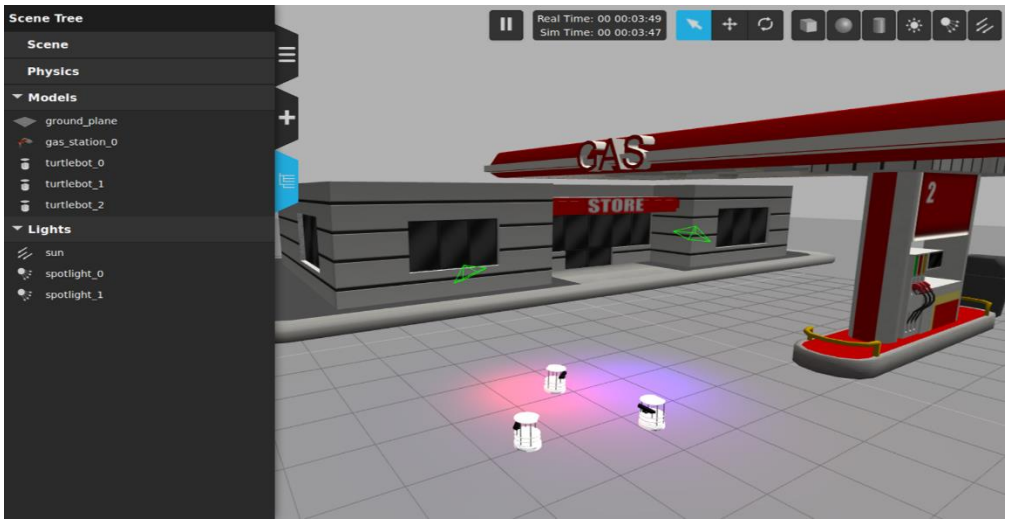

Figura 3.9 Escenario Creado en Gazebo

JdeRobot ha desarrollado varios plugins usando Gazebo, el que nos interesa en este proyecto es el que nos permite manejar a través de él un cuadricoptero, en concreto el modelo ARDrone de Parrot. Este plugin facilita la programación a más bajo nivel como son las conexiones de forma que podemos manejar el dron en un mundo creado por Gazebo donde podremos depurar nuestro código.

INTRODUCIR CAPTURA DEL MUNDO DE GAZEBO

3.3.2 Actualización JdeRobot

DroneWebRTC usaba la versión 5.3 de JdeRobot junto con la versión 5 de Gazebo, todo ello bajo el sistema operativo Linux Ubuntu 14.04. 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 que han afectado a este proyecto.

- Cambio de SO, la nueva versión de JdeRobot es soporatada 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 de Gazebo.
- 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 asi 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.

En cuanto a los componentes usados también se ha producido un cambio en la configuración de ArDroneServer, como se comentó en el anterior apartado con este plugin permitía la conexión con un ArDrone a través de seis conexiones que pertenecen a seis interfaces ICE conectados mediante websockets. Tras el cambio de versión se crea una única conexión mediante websockets que nos permite manipular los seis interfaces.