ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA
Ingeniería del Software

Teleoperación de un robot móvil mediante dispositivos Android

Teleoperation of a mobile robot using Android devices

Realizado por
JESÚS RAMÍREZ RAMOS
Tutorizado por
Dr. VICENTE M. ARÉVALO ESPEJO
Dr. A. JAVIER GONZÁLEZ JIMÉNEZ

Departamento
Ingeniería de Sistemas y Automática

UNIVERSIDAD DE MÁLAGA
MÁLAGA, Febrero 2016

Fecha defensa:
El Secretario del Tribunal

Resumen: En el siguiente trabajo se aborda un problema para solventar la comunicación con los robots del departamento MAPIR de la Universidad de Málaga, anteriormente sólo podían ser teleoperados mediante comandos escritos en Skype, así que se procede a diseñar un cliente móvil para Android que nos permite conectarse en tiempo real a un robot, obtener la imagen de lo que su cámara capta y además permitir su teleoperación. Por su parte, el robot corre un servidor que administra esos datos al cliente para trabajar conjuntamente. Dicho trabajo se desarrolla haciendo uso de nuevas tecnologias y protocolos como es WebRTC (de Google) para el intercambio de imágenes y del lado del servidor, se ha usado NodeJS.

Palabras claves: video, streaming, webrtc, nodejs, robotica, teleoperacion, android, robots

Abstract: Throughout this project, we will try to solve some shortcomings of MAPIR research group at the University of Málaga. So far, they have worked with Skype to communicate with other robots, so we have designed an Android mobile client that allows teleoperation in realtime. All this research is done by using new technologies such as WebRTC, NodeJS and Python.

Keywords: video, streaming, webrtc, nodejs, robotics, remote, remotecontrol, android, robot

Índice de Contenidos

1 Introducción 15
1.1 Contexto 15
1.2 Objetivos Generales 17
1.3 Medios Materiales 18
1.4 Estructura del documento 19
2 Descripción de la problemática 21
2.1 Introducción 21
2.2 Soluciones similares 22
3 Solución propuesta 25
3.1 Solución Técnica 25
3.2 OpenMora 26
3.2.1 MOOS 26
3.2.2 MRPT 27
3.3 WebRTC 28
3.3.1 WebRTC en el navegador 29
3.3.2 Peligro en las comunicaciones 29
3.3.3 Seguridad en WebRTC 30
3.3.4 Conclusión sobre WebRTC 30
3.4 NodeJS 31
3.4.1 Módulos 31
3.4.2 Comunidad 32
3.5 Android 33
4 Desarrollo de la solución 35
4.1 Servidor NodeJS “Servidor-Robot” 35
4.2 Cliente WebRTC "Cliente Móvil Android" 38
4.3 Librerías externas 42
4.4 Permisos en Android 43
4.5 Diagramas de clases de la aplicación Android 44
4.6 Diseño de la vista Cliente Android 54
5 Pruebas de Rendimiento: SKYPE VS WEBRTC 61

6 Conclusiones y futuras mejoras 65
6.1 Conclusiones finales 65
6.2 Futuras líneas de trabajo 66
7 Referencias 67
8 Apéndices 71
8.1 Apéndice I: Código Fuente 71
8.2 Apéndice II: Manual de usuario 73
8.3 Apéndice III : Funcionamiento de WebRTC 81
8.4 Apéndice IV: Conceptos básicos para NodeJS 91
8.5 Apéndice V: Detección de errores con Splunk Mint 95

Índice de Figuras

1.1.1: Robots del grupo de investigación MAPIR perteneciente al departamento de Ingeniería de Sistemas y Automática de la Universidad de Málaga [1]. 16
2.1.1: Logotipo y breve explicación sobre OpenMORA [2]. 23
3.3.1: Logo corporativo de WebRTC [3]. 28
3.3.1: Comunicación en tiempo real que se produce en un navegador [4]. 29
3.4.2: Empresas colaboradoras [5]. 32
3.5.1: Características de Android: Fuente Wikipedia [6]. 33
3.5.2: Cuota de Mercado por Sistema Operativo [7]. 34
4.1.1: Visión del pequeño servidor NodeJS en su primera fase [8]. 37
4.1.2: Visión de la imagen obtenida a través de la cámara web [9]. 37
4.2.1: 15.7.2014 Audio y Video retransmitido al teléfono [10]. 38
4.2.2: 16.7.2014 Círculos pintados encima del contenedor de video [11]. 39
4.2.3: 4.08.2014 Joystick semi-funcional [12]. 40
4.2.4: Marzo 2015 Perfeccionamiento del interfaz de las preferencias [13]. 41
4.5.1: Diagrama de clases Completo – Simplificado [14]. 44
4.4.2: Diagrama de Clases WebRTC [15]. 45
4.4.3: Diagrama de Clases MainActivity [16]. 47
4.4.4: Diagrama de Clases de VideoStreamView [17]. 49
4.4.5: Diagrama de Clases Joystick [18]. 51
4.4.6: Diagrama de Clases Versión Final [19]. 54
4.5.1: Detalle sobre los componentes en la vista [20]. 54
4.5.3: Seleccionando todos los compontes del diseño Tablet [21]. 58
4.5.4: Seleccionando todos los compontes del diseño en vista teléfono [22]. 58
4.5.7: Diseño de la vista del menú. [23]. 59
4.5.8: Diseño de la vista del menú en un tablet. [24]. 59
5.1.1: Comparativa de ancho de banda WebRTC vs Skype [25] 62
8.1.1: Mi repositorio de Github con ambos proyectos [26]. 71
8.3.1: Cliente –Servidor WebRTC [27]. 84
8.3.2: Arquitectura WebRTC [28]. 85
8.3.3: Encontrando candidatos (Proceso de encontrar interfaces de red) [29]. 87
8.3.4: Flujo de datos en WebRTC [30]. 87
8.3.5: Arquitectura de la aplicación de video chat de Google appRTC [31]. 88
8.3.6: La API del canal de Google: Estableciendo un canal [32]. 89
8.5.1: Ventana de la Web MINT para la depuración de la aplicación móvil [33]. 96
8.5.2: Diversos errores capturados a lo largo de 90 días [34]. 97

AGRADECIMENTOS

Me gustaría agradecer en primera instancia a Vicente Arévalo Espejo, por permitirme desarrollar una idea mía propia, que nació de las visitas realizadas al departamento de Robótica de la universidad de Málaga (MAPIR Ingeniería de Sistemas y Automática) en la que trabaja un gran amigo y vecino; José Raúl Ruiz Sarmiento, el cual, al mostrarme parte del trabajo que realizan en diferentes robots, hizo surgir en mí la idea de suplir algunas necesidades que no estaban cubiertas, cómo la teleoperación de los diferentes Robots del departamento desde dispositivos móviles.

También no puedo olvidarme de Javier González Jiménez que no dudó en apuntarse a la idea y tutorizar junto a Vicente dicho proyecto.

Por supuesto también a todo el equipo de MAPIR, que tanto me ha ayudado y aconsejado en mis visitas a los laboratorios para trastear con los diferentes robots, Curro, Javi, Carlos, Rubén, etc...

Quisiera además dar las gracias a todas las personas que me han acompañado en esta etapa en la universidad, haciendo de ella una etapa inolvidable, en especial: Nahuel, Edu, Fabián, Ale, Pepe y un largo etc. que no cabrían en esta sección; a todos ellos por los trabajos en grupo, horas de estudio, madrugones, desayunos, risas y noches de ocio.

Por ultimo no quiero olvidarme de mi familia, mi hermana Beatriz y mis padres Mª del Carmen y Antonio por ayudarme en todo lo que ha estado en sus manos para que consiguiera llegar hasta aquí.

A todos, muchísimas gracias.

Capítulo 1

1 Introducción

1.1 Contexto

Desde el nacimiento de Internet siempre se pensó en crear una gran red que comunicara distintos dispositivos, siendo posible conectar máquinas que se encontraban en diferentes ciudades, hasta la revolución tecnológica que estamos viviendo, donde cada día existen más aparatos conectados a la red.

Este “boom” de la conectividad apareció primero con los ordenadores personales (allá por 1981 IBM hizo la presentación de los primeros equipos destinados al uso personal) pero no sería hasta 1990 cuando la gran red de redes fue accesible al gran público, cuando comenzaría la carrera que sigue su curso a día de hoy. Teléfonos *smartphones*, tabletas, televisiones, lavadoras, frigoríficos, coches… cada vez más artículos de uso doméstico están teniendo conectividad con la gran red que es Internet. Ahora se llama el internet de las cosas (*IoT Internet of Things*).

Las relaciones sociales también se han visto afectadas por estos cambios, existiendo multitud de redes sociales, chats, foros, para intercambiar información. Podemos poner por ejemplo como ahora la televisión se debate en la red mediante la red social Twitter con el uso de los *hashtag*, donde los espectadores pueden dar su opinión en tiempo real sobre los eventos que acontecen en diversos programas. Sobre el uso de las comunicaciones de vídeo, contar como anécdota como surgieron. En el departamento de Informática de la Universidad de Cambridge existía una cafetera en un sótano. Si alguien quería un café tenía que bajar desde su despacho y, si lo había, servirse una taza. Si no lo había, tenía que prepararlo. Las normas decían que el que termina la cafetera debe rellenarla, pero siempre había listos que no cumplían las reglas.

En 1991, Quentin Stafford-Fraser y Paul Jardetzky, que compartían despacho, hartos de bajar tres plantas y encontrarse la cafetera vacía decidieron pasar al contraataque. Diseñaron un protocolo cliente-servidor que, conectándolo a una cámara, trasmitía una imagen de la cafetera a una resolución de 128x128 pixels. Así, desde la pantalla de su ordenador sabían cuándo era el momento propicio para bajar por un café, y de paso sabían quiénes eran los que terminaban la cafetera y no la volvían a llenar. El protocolo se llamó XCoffee y tras unos meses de depuración se decidieron a comercializarlo. En 1992 salió a la venta la primera cámara web llamada XCam.

Con esta breve presentación queremos poner de manifiesto cómo de una idea tan sencilla, se convirtió posteriormente en un cambio global, algo así está ocurriendo en el ámbito de la robótica.

Desde los años sesenta, la robótica está avanzando a un ritmo vertiginoso. Cuando apareció el primer robot industrial en 1961, su uso ha ido incrementándose en todas las áreas de la sociedad, tanto para tareas automatizadas, como en actividades lúdicas. Ahora, se están empezando a comercializar para los hogares, realizando tareas cotidianas como limpieza de la casa, etc. Aunque aún son solo pequeños asistentes que realizan tareas muy determinadas, en un futuro, no muy lejano se espera que dichos robots domésticos puedan realizar múltiples y complejas actividades en nuestros hogares; realizar la compra vía internet, poner lavadoras, recoger la ropa, hacer la cama, cocinar, cuidar la casa, etc.

Llegado este punto se hace imprescindible el desarrollo de interfaces de usuario que faciliten la comunicación con estos robots. Unos interfaces flexibles e intuitivos que permitan la programación y/o teleoperación de estos dispositivos a personal no experto. Es aquí donde los dispositivos móviles actuales, smartphones y tabletas, cobran especial interés. Si se tiene en cuenta que en la actualidad estos dispositivos están cada vez más presentes en nuestros hogares, es natural pensar en soluciones que permitan comunicarnos con estos robots.

De la necesidad de una vía de comunicación con el entorno de los robots, surge el uso del *streaming*. Streaming o videoconferencia es la comunicación simultánea bidireccional de audio y vídeo, que permite mantener reuniones con grupos de personas situadas en lugares alejados entre sí.

Giraff *Sancho* *Rhodon*

Figura 1.1.1: *Robots del grupo de investigación MAPIR perteneciente al departamento de Ingeniería de Sistemas y Automática de la Universidad de Málaga [1].*

1.2 Objetivos Generales

El objetivo de este proyecto es desarrollar una aplicación para la plataforma Android que permite a una persona, desde su dispositivo móvil Android, acceder remotamente a un robot y controlarlo. Este objetivo se puede descomponer a su vez en los siguientes objetivos particulares:

- Desarrollar los módulos de software necesarios que permitan conectar de forma eficiente una aplicación Android con la arquitectura robótica utilizadas por los robots del grupo de investigación *MAPIR* (**M**Achine **P**erception and **I**ntelligent **R**obotics)**.**

- Diseñar e implementar una interfaz gráfica intuitiva y amigable que permita, entre otras tareas acceder a los sensores y actuadores de la base robótica y operarlos de forma remota.

- Validar y probar la aplicación desarrollada en uno de los robots del grupo.

MAPIR es un grupo de investigación perteneciente al Dpto. de Ingeniería de Sistemas y Automática de la Universidad de Málaga, y, tal y como nos dice su nombre, entre sus áreas de interés se encuentra la percepción e inteligencia de robots.

En este proyecto se usan conocimientos técnicos adquiridos durante la carrera, así como los conseguidos en la investigación del presente proyecto.

Destacar que, actualmente, el acceso a los recursos del robot, tanto de la cámara como de otros sensores, no puede realizarse de forma directa, sino que se realiza a través de *OpenMORA*. *OpenMORA* es una arquitectura distribuida basada en *MOOS* (software de comunicaciones desarrollado en C++ para la investigación en robótica) desarrollada por el grupo de investigación MAPIR y que se utiliza actualmente en todos los robots del grupo. Actualmente está abierta al público, aunque continua en fase experimental. Sobre dichos términos nos centraremos en posteriores capítulos de la memoria.

La complejidad requerida en este proyecto pasa por integrar en una aplicación móvil de forma nativa, una interfaz lo más minimalista, que permite el visionado de vídeo y envía comandos de control al robot (en este caso control remoto/ teleoperación). Por ello, se han estudiado las diferentes posibilidades y las plataformas existentes que se verá en capítulos posteriores de la memoria.

1.3 Medios Materiales

En la realización del proyecto se han usado los siguientes elementos hardware y software:

Hardware:

1. PC Sobremesa con Sistema Operativo Windows 10.
2. Ordenador Portátil con Sistema Operativo OSX 10.11 El Capitán.
3. Teléfono Móvil (Samsung N7100 con Android 6.0).
4. Teléfono Móvil (Nokia X con Android 4.1.2).
5. Tableta (Novo7 Basic con Android 4.4).
6. Plataforma robótica del Grupo de Investigación de Percepción y Robótica Inteligente (MAPIR).

Software:

1. Android Studio 2.0 Preview 7.
2. Visual Paradigm.
3. Microsoft Visual Studio 2013.
4. Eclipse IDE.
5. Arquitectura robótica OpenMORA.
6. Librería de software robótico MRPT (The Mobile Robot Programming Toolkit).
7. Python 3.5.0.
8. NodeJS 5.1.0 x64.
9. Google Chrome 46.

1.4 Estructura del documento

La presente memoria consta de diferentes capítulos:

- En el segundo capítulo, se narran algunos aspectos de la problemática, así como otras posibles alternativas y el porqué de las decisiones de diseño adoptadas finalmente.

- La solución técnica se detalla en el tercer capítulo. En él se exponen las diferentes características de cada tecnología elegida, además se argumenta el uso de las herramientas empleadas.

- El desarrollo de la solución se aborda en el cuarto capítulo. Se explican los pasos a seguir para alcanzar la solución final, explicando detenidamente cómo se implementaron clases para el posterior desarrollo, así como el diseño de la interfaz.

- En el quinto capítulo se discuten los resultados obtenidos y se presenta una comparativa con la solución hasta entonces usada.

- Por último, se realiza un breve balance de las conclusiones finales en el sexto capítulo. De este modo se analizarán los objetivos iniciales planteados junto con los resultados obtenidos. Además, se comentarán posibles líneas de trabajo futuras para la ampliación de este proyecto.

- Al final de este proyecto se incluye un apéndice con información adicional:

    - Código Fuente.
    - Manual de usuario (Instalación en servidor y cliente Android).
    - Funcionamiento de WebRTC.
    - Conceptos básicos sobre NodeJS.
    - Detección de errores mediante la herramienta Splunk Mint.

Capítulo 2

2 Descripción de la problemática

El problema tiene con una complejidad alta, debido a que interfieren diferentes sistemas en tiempo real debido a que no sólo se trata de crear una conexión de vídeo con dos dispositivos simultáneamente, sino que también debe permitirnos teleoperar una de las partes; haciéndolo con el menor retardo posible y mejorando la solución usada hasta ahora.

2.1 Introducción

A lo largo de este capítulo se expondrán las dificultades y cuestiones que se plantean en la actualidad cuando se habla de *streaming*, que han originado el inicio de este proyecto.

El núcleo tecnológico usado en un sistema de videoconferencia es la compresión digital de los flujos de audio y vídeo en tiempo real. Su implementación proporciona importantes beneficios, como el trabajo colaborativo entre personas geográficamente distantes y una mayor integración entre grupos de trabajo.

Actualmente, las limitaciones técnicas, tales como el sonido deficiente, la mala calidad de las imágenes, la poca fiabilidad, la complejidad y el costo, han quedado atrás dando lugar a videoconferencias de alta calidad con audio, vídeo, transferencia de archivos y de un costo más que accesible a la mayoría de los interesados.

Es decir, la videoconferencia ofrece una solución accesible a la necesidad de comunicación, con sistemas que permiten el transmitir y recibir información visual y sonora entre puntos o zonas diferentes evitando así los gastos y pérdida de tiempo que implican el traslado físico de la persona. Estas ventajas hacen de la videoconferencia el segmento de mayor crecimiento en el área de las telecomunicaciones.

Los robots del grupo de investigación MAPIR hacen uso del protocolo de video Skype realizando llamadas desde dicha aplicación, lo cual requiere que la aplicación este instalada en el robot, así como en el cliente que se conecta a él, siendo esta una solución privada y propietaria de Microsoft. También debemos tener en cuenta que se envía texto por el chat de Skype para enviar órdenes a dichos robots.

Para comunicarnos con los robots, y acceder a su hardware, se utiliza una arquitectura robótica denominada OpenMora, mantenida por la Universidad de Málaga (Grupo de investigación MAPIR) y el Laboratorio de Ingeniería Mecánica de la universidad de Almería, además de otros colaboradores individuales.

A continuación, se detallan diversas soluciones disponibles en la actualidad y que han sido analizadas al inicio de este proyecto.

2.2 Soluciones similares

La teleoperación de robots requiere de una comunicación fluida entre robot y operario de tal forma que permita realizar las operaciones de manera eficiente. Esto nos obliga a implementar un mecanismo que permita transferir vídeo y audio entre plataformas cliente y servidor con una tasa de transferencia lo suficientemente buena como para poder trabajar en "tiempo real" (esto es, sin retardo).

Durante el proceso de investigación hemos encontrado diversas soluciones que cubrían desde elaborar una solución nativa que conectara servidor (en este caso robots), cliente (dispositivos móviles) a soluciones comerciales.

En primer caso, evaluamos cómo funcionaba actualmente el sistema, pensado para equipos de sobremesa, era necesario tener la aplicación Skype instalada en los equipos y para crear un cliente para dispositivos móviles, dependíamos de la API de Skype, en este caso inexistente. Por lo que desde el principio ya se sabía que había que buscar otra solución, ya que no era viable instalar Skype en cada dispositivo móvil y enviar posiciones de movimiento mediante escritura en la ventana de chat.

También se evaluaron si existían soluciones comerciales para nuestro problema y aunque encontramos que existían algunas como Stäubli Robotics Suite (http://www.staubli.com/es/robotics/software-para-robots/staeubli-robotics-suite/) y Kuka-Robotics (http://www.kuka-robotics.com/usa/en/products/software/remotecontrol/). Una vez contactados con dichos departamentos comerciales, tanto el precio de las licencias, con un alto coste inviable, cómo un problema aún mayor: No ofrecían soporte para OpenMora por lo que era necesario tener acceso hardware a los distintos periféricos de cada robot, con lo que esto dejaba inoperativa la arquitectura en pizarra diseñada por el grupo de investigación MAPIR. Dichas soluciones también fueron descartadas.

Se evaluó si era posible acceder al hardware nativamente para enviar las imágenes capturadas de sus cámaras a un cliente móvil. Aunque esta solución era en un principio factible, fue descartada por su complejidad y porque estaba violando el principio de OpenMORA al igual que las soluciones comerciales, así que para conseguir una solución viable debía de respetarse la arquitectura usada en todos los robots del grupo de investigación.

Figura 2.1.1: *Logotipo y breve explicación sobre OpenMORA [2].*

Es por esto que la solución debía centrarse en ser compatible con OpenMORA para que fuese fácilmente portable a otros equipos del grupo de investigación.

Quizás un servidor de aplicaciones nos permitía la portabilidad y el acceso a la arquitectura en pizarra usada por OpenMORA. Aquí es donde por primera vez aparece *NodeJS* [3], un puro ejemplo de un servidor de aplicaciones modular, que trabaja de forma muy eficiente gracias a su planificador asíncrono y lo más importante, con una posible solución, gracias a su gran comunidad; en este caso me refiero al módulo *MQTT* [4] (https://www.npmjs.com/package/mqtt), el cual permitiría la conexión con openMora, que trabaja haciendo uso de un *broker* mediante el protocolo de transmisión de datos MQTT.

La arquitectura OpenMORA (basada a su vez en *MOOS*) consta de múltiples elementos funcionales, denominados “agentes”, y un instrumento de control denominado “pizarra”. Los agentes suelen estar especializados en una tarea concreta o elemental. Todos ellos cooperan para alcanzar una meta común, si bien, sus objetivos individuales no están aparentemente coordinados.
El comportamiento básico de cualquier agente consiste en examinar la pizarra, realizar su tarea y escribir sus resultados en la misma pizarra. De esta manera, otro agente puede trabajar sobre los resultados generados por otro para realizar su propio trabajo. El procesamiento (y el intercambio de información) termina cuando se alcanza el objetivo marcado a través de los resultados escritos en la pizarra por los distintos módulos.

Recordemos que el *acceso al hardware es controlado* por OpenMORA.

Las demás soluciones sólo dividían el problema en distintos pero aumentando su complejidad en vez de simplificarla , así que se ha diseñado una solución que está adaptada a las necesidades del grupo de investigación MAPIR de la Universidad de Málaga, siguiendo siempre las líneas y el feedback de los compañeros del laboratorio.

Capítulo 3

3 Solución propuesta

3.1 Solución Técnica

Nuestra solución pasa por crear un cliente (para dispositivos móviles) y un servidor (compatible con todos los robots del grupo de investigación), que permitan el envío/recepción de la señal de video, así como el envío/recepción de comandos.

Durante esta sección se van a describir las tecnologías utilizadas para llegar a generar la solución, aportando información sobre cada una de ellas, así como una muestra de su funcionamiento.

En este caso viendo el auge de las tecnologías web, se ha optado por el uso de una tecnología de código abierto, llamada *WebRTC* (RTC **R**eal **T**ime **C**ommunication), compatible con la mayoría de navegadores web y que permite de forma fácil y muy eficiente hacer conexiones peer-to-peer entre dos clientes. Además, dicha plataforma nos permite crear en el mismo canal una vertiente para el envío de información llamado DataChannel, con el cual podemos aprovechar la conexión que se realiza para él envío de vídeo, para también, enviar los comandos de teleoperación (a fecha Julio 2014 dicha función DataChannel aún no está del todo operativa). Todo esto va a ser desarrollado de forma nativa en dispositivos móviles con sistema operativo *Android*.

En la parte del servidor nos hemos decantado por hacer uso de la tecnología NodeJS. Cómo se comentó en el capítulo anterior, era la opción más viable debido a que cumple el requisito fundamental, tener soporte para MQTT además de ser una tecnología que continua también en desarrollo y tiene una gran comunidad detrás, muy rápida, fácilmente escalable y multiplataforma, por lo que el servidor Web que vamos a crear en NodeJS será funcional tanto en Windows, OSX, y distribuciones de Linux.

3.2 OpenMora

OpenMora (Open Mobile Robot Aquitecture) es una arquitectura para robots móviles desarrollada por el grupo MAPIR. La arquitectura OpenMora está basada en MOOS y MRPT. Esta implementa un diseño modular, en el cual, cada módulo cuenta con la característica de realizar un único objetivo. Esta arquitectura ofrece diferentes módulos para la localización, uso de sensores, almacenamiento de datos, navegación reactiva, navegación planificada, etc.

Para poder entender mejor el funcionamiento de la arquitectura OpenMora, a continuación, damos una visión general de MOOS y de la librería de la MRPT.

3.2.1 MOOS

MOOS (Mission Oriented Operating Suite) es un software de comunicaciones desarrollado en C++ para la investigación en robótica.
Para poder entender su funcionamiento, se ha de pensar en un servidor al cual se conectan todos sus clientes, en el que cada cliente cuenta con un objetivo. El cliente puede publicar datos en el servidor o bien leer datos que otros clientes hayan publicado en él. Es decir, un cliente puede publicar un dato que otro u otros clientes más tarde leerán. De esta forma se realiza una comunicación entre todos los módulos de forma separada sin la interactuación directa de unos módulos con otros.
Como se ha comentado, cada módulo se encarga de una tarea específica dentro del robot. MOOS cuenta con una topología en estrella. La capa fundamental del núcleo de MOOS se trata de la capa de comunicaciones basada en TCP/IP. Cada aplicación desarrollada dentro de MOOS establece una conexión a un servidor central, denominado MOOSDB\MOOS Database.

Todas las comunicaciones que se establecen entre diferentes aplicaciones pasan a través del servidor central (MOOSDB). La red establecida entre las aplicaciones y el servidor central cuentan con las siguientes características:

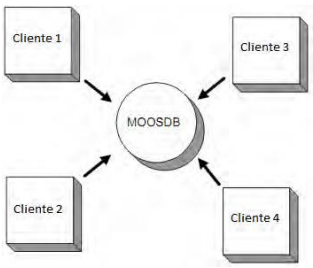


Figura 3.2.1: *La topología seguida por MOOS es una topología en estrella, en la cual cada cliente cuenta con un único canal de comunicación con el servidor MOOSDB*.

- La comunicación entre aplicaciones no se establece directamente entre ellas, no se trata de una comunicación punto a punto. Estas comunicaciones se realizan todas a través del servidor MOOSDB.

- Todas las comunicaciones entre la aplicación cliente desarrollada y el servidor (MOOSDB) son llevadas a cabo por el cliente. Es decir, el servidor (MOOSDB) no intenta establecer comunicación con sus clientes, es el cliente el que se conecta al servidor.

- Cada cliente cuenta con un nombre único el cual lo identifica desde el punto de vista del servidor. Esta característica es importante tenerla en cuenta a la hora de desarrollar nuevas aplicaciones, para que estas no cuenten con nombres duplicados.

- Cada cliente, no sabe de la existencia de otros clientes.

- Un cliente no le puede enviar información directa a otro cliente, dicha información solo puede ser enviada al servidor MOOSDB.

- La red formada puede estar distribuida sobre diferentes maquinas corriendo todas estas sobre diferentes sistemas operativos, las cuales puedan soportar MOOS.

3.2.2 MRPT

MRPT (Mobile Robot Programming Toolkit) es un proyecto desarrollado y mantenido por el grupo MAPIR de la Universidad de Málaga que persigue la creación de módulos a través de librerías para facilitar el desarrollo de aplicaciones software, no sólo en el ámbito de la robótica móvil como su nombre apunta, sino también aplicaciones científicas. Estas librerías incluidas en MRPT incluyen herramientas, para, por ejemplo, trabajar con matrices, vectores, representaciones gráficas, etc.

Si nos centramos en el campo de la robótica, MRPT proporciona implementaciones muy eficientes de algoritmos para SLAM (“Simultaneous Localization and Mapping”), visión por computador, etc. Las librerías desarrolladas incluyen clases para utilizar la representación de escenas 3D, funciones de probabilidad y densidad, inferencia bayesiana, procesamiento de imágenes, etc.

3.3 WebRTC

WebRTC es un códec multimedia dentro del navegador que permite a cualquier desarrollador Web construir una aplicación JavaScript de forma sencilla. Permite crear comunicaciones VoIP de forma sencilla a la web, siendo fácilmente portables a cualquier plataforma y es de código abierto.

***Figura 3.3.1**: Logo corporativo de WebRTC [3].*

Debemos tener en cuenta que el signado de las señales no está incluido en WebRTC, por lo que cada uno puede decidir si utilizar una señal propietaria o un estándar (se hablará más delante de la solución elegida).

WebRTC es un nuevo frente abierto en una guerra por las aplicaciones Web en la industria tecnológica. Comunicación en tiempo real sin librerías externas, desde el navegador. Imagina un mundo donde tu teléfono, televisión y ordenador pueden comunicarse a través de un canal común de datos. Imagina que es tan fácil añadir video, intercambio de ficheros, etc. Esa es la idea por la que surge WebRTC.

WebRTC es un proyecto de Google soportado por un amplio número de navegadores (aprobados por el estándar W3C), para visualizar un ejemplo de cómo ha implementado dicho protocolo Google, su lugar de pruebas está disponible en apprtc.appspot.com

Uno de los mayores logros para la web es permitir la interacción humana vía audio/video en tiempo real, por lo que WebRTC debe ser natural en una aplicación web como si de un campo de texto en un formulario se tratara.

Sin él, estamos limitando nuestra habilidad para innovar y diseñar nuevas formas de interactuar entre las personas.

3.3.1 WebRTC en el navegador

Una aplicación WebRTC (típicamente escrita como una mezcla de código html con JavaScript) interactúa entre varios navegadores a través de la API estandarizada WebRTC permitiendo explotar los recursos multimedia de un equipo en tiempo real, por supuesto dichas capacidades vendrán dadas por la capacidad del caudal de banda, versión y tipo de navegador, etc.

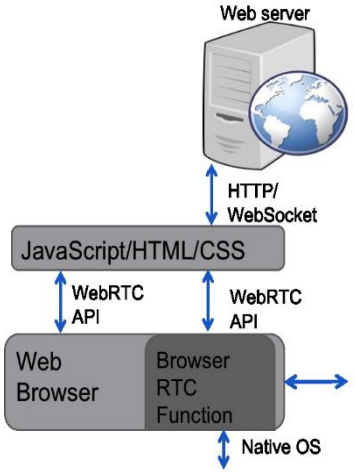


Figura 3.3.1: *Comunicación en tiempo real que se produce en un navegador [4].*

3.3.2 Peligro en las comunicaciones

Hay varias formas en las que una comunicación en tiempo real o aplicación puede comprometerse en aspectos relacionados con la seguridad. Por ejemplo:

- Datos multimedia o datos sin cifrar, pueden ser interceptados en la ruta mediante navegadores, o mediante la conexión entre servidor- navegador.

- Una aplicación puede grabar y distribuir video o audio sin que el usuario tenga constancia de ello.

- Malware o virus que pueden instalarse en el sistema, haciendo pasar por inocuos programas y/o librerías.

3.3.3 Seguridad en WebRTC

WebRTC tiene bastantes mejoras para evitar este tipo de problemas:

- WebRTC implementa el uso de protocolos seguros como “DTLS” y “SRTP”.

- El cifrado es algo obligatorio para todos los componentes de WebRTC incluyendo las señales de control.

- WebRTC no es una librería: Es un componente integrado en el navegador en una *sandbox* y no en un proceso independiente, dichos componentes no requieren una instalación y son actualizados cuando el navegador es actualizado.

- El acceso al micrófono y la cámara deben ser pertinentemente aprobados para ser usados, además de ser claramente mostrados en la interfaz de usuario.

3.3.4 Conclusión sobre WebRTC

La API y el estándar de WebRTC puede democratizarse y descentralizar todas las herramientas para la creación de contenido. Por ejemplo, para telefonía, juegos, producción de video, creación de música y muchas otras aplicaciones.

En un apartado más adelante se abordará una comparativa sobre el gasto de ancho de banda entre usar Skype y WebRTC, porque WebRTC tiene una ventaja bastante importante a tener en cuenta.

El uso de WebRTC ha crecido exponencialmente en los últimos meses y continúa creciendo, se espera que todas las comunicaciones vía voz-imagen sean bajo dicho protocolo y que esté se establezca como el estándar en la web, reemplazando a aplicaciones de terceros y *plugins* externos.

Para más información sobre la organización de WebRTC, puede visualizarse en el apéndice correspondiente a WebRTC (Cómo está organizado el protocolo y la pila de llamadas [7]).

3.4 NodeJS

Node.js es un entorno de programación en la capa del servidor (pero no limitándose a ello) basado en el lenguaje de programación ECMAScript, asíncrono, con I/O de datos en una arquitectura orientada a eventos y basado en el motor V8 de Google. Fue creado con el enfoque de ser útil en la creación de programas de red altamente escalables, como por ejemplo, servidores web.

Node.js es similar en su propósito a Twisted o Tornado de Python, Perl Object Environment de Perl, React de PHP, *libevent* o *libev* de C, EventMachine de Ruby, *al contrario que la mayoría del código JavaScript, no se ejecuta en un navegador, sino en el servidor.* Node.js implementa algunas especificaciones de CommonJS. Además, incluye un entorno REPL para depuración interactiva muy poderosa.

Versión de un hola mundo de un Servidor HTTP escrito en Node.js:

```
var http = require('http');

http.createServer(function (request, response) {

   response.writeHead(200, {'Content-Type': 'text/plain'});

   response.end('Hello World\n');

}).listen(8000);

console.log('Server running at http://127.0.0.1:8000/');
```

3.4.1 Módulos

NodeJS incorpora varios "módulos básicos" compilados en el propio binario, como por ejemplo el módulo de red, que proporciona una capa para programación de red asíncrona y otros módulos fundamentales, como por ejemplo *Path, FileSystem, Buffer, Timers* y el de propósito más general *Stream*. Es posible utilizar módulos desarrollados por terceros, ya sea como archivos ".node" precompilados, o como archivos en *javascript* plano. Los módulos Javascript se implementan siguiendo la especificación *CommonJS* para módulos, utilizando una variable de exportación para dar a estos scripts acceso a funciones y variables implementadas por los módulos.

Los módulos de terceros pueden extender node.js o añadir un nivel de abstracción, implementando varias utilidades middleware para utilizar en aplicaciones web, como por ejemplo los frameworks *connect* y *express*. Pese a que los módulos pueden instalarse como archivos simples, normalmente se instalan utilizando el *Node Package Manager* (npm) que nos facilitará la compilación, instalación y actualización de módulos, así como la gestión de las dependencias. Además, los módulos que no se instalen el directorio por defecto de módulos de Node necesitarán la utilización de una ruta relativa para poder encontrarlos. En la *wiki* NodeJS proporciona una lista de varios de los módulos de terceros disponibles.

3.4.2 Comunidad

Existe una comunidad muy activa de desarrolladores de Node.js que se comunican a través de grupos de discusión, *nodejs* y *nodejs-dev* y el canal IRC #node.js en *freenode*. La comunidad se reúne en NodeConf, una convención de desarrolladores centrada en NodeJS.

Alguna de las empresas más famosas que ya se han migrado hacia los servicios que ofrece la plataforma NodeJS son: Microsoft, Paypal, Uber, Yahoo, LinkedIn, The New York Times, Ebay.

Puedes ver una lista de las principales empresas que utilizan Node.js desde su web: https://nodejs.org/en/foundation/members/.

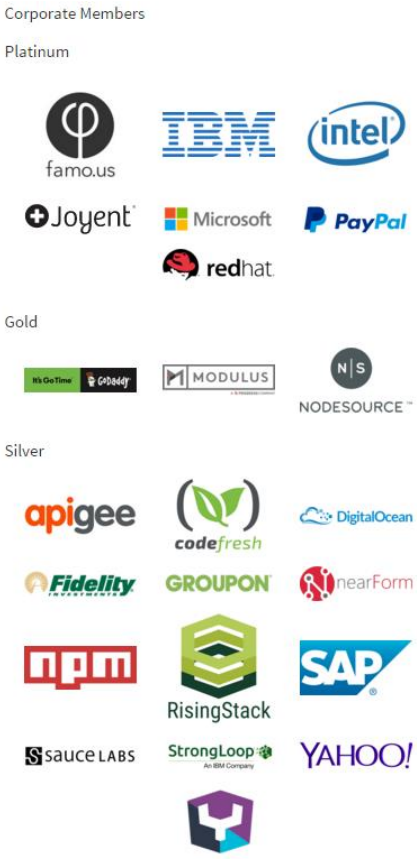


Figura 3.4.2: *Empresas colaboradoras [5]*

Por ejemplo, LinkedIn utiliza NodeJS en combinación con HTML5 en su aplicación móvil para dotarla del máximo de rapidez posible.

3.5 Android

Es un sistema operativo basado en el núcleo de Linux (ver figura 3.5.1). Fue principalmente diseñado para dispositivos móviles con pantalla táctil, ya sean teléfonos inteligentes, tabletas, relojes, televisores, automóviles, etc. Originalmente fue desarrollada por la empresa Android Inc. la cuál fue adquirida por Google en julio de 2005. Entonces el desarrollo continuo de la mano de Google y de la Open Handset Alliance (un conjunto de fabricantes y desarrolladores de hardware/software y operadores de telefonía).

El primer dispositivo que apareció en el mercado usando Android fue el HTC Dream y se vendió a partir de octubre de 2008.

Las ventas de terminales Android, suman más que las combinadas de Windows Phone y IOS juntas.

Características de la arquitectura Android:

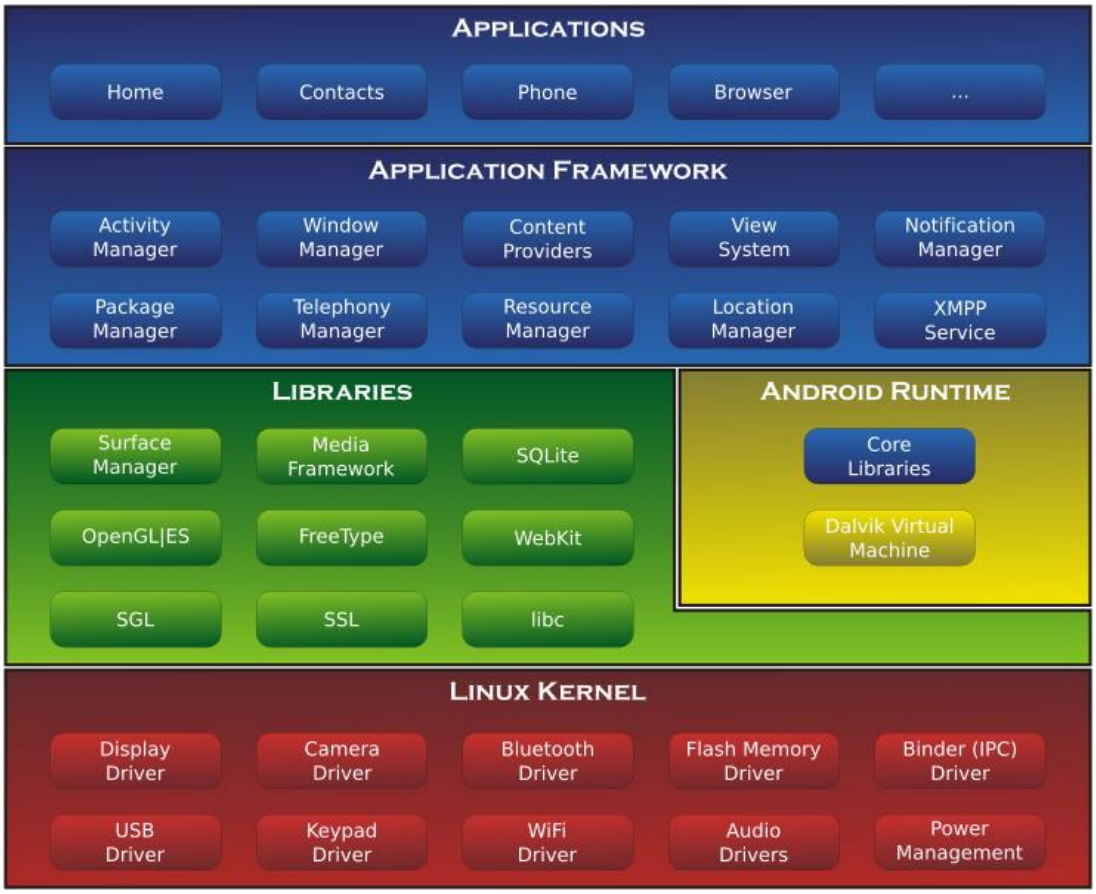


Figura 3.5.1: *Características de Android: Fuente Wikipedia [6].*

En lo referente a porque se ha elegido Android frente a otras soluciones, sólo hay que mirar estadísticas y es fácil reconocer el número de personas en el mundo que usan un dispositivo Android.

En los siguientes gráficos tenemos una información que nos da una idea de cómo está el mercado actualmente:

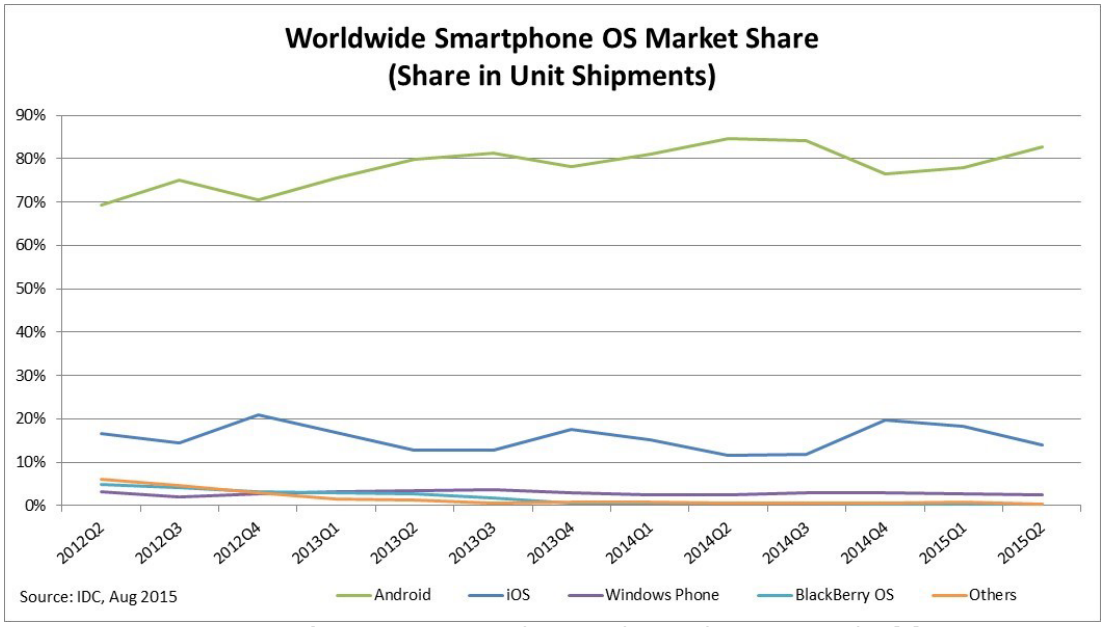


Figura 3.5.2: *Cuota de Mercado por Sistema Operativo [7].*

Período	Android	iOS	Windows Phone	BlackBerry OS	Others
Q2 2015	82.8%	13.9%	2.6%	0.3%	0.4%
Q2 2014	84.8%	11.6%	2.5%	0.5%	0.7%
Q2 2013	79.8%	12.9%	3.4%	2.8%	1.2%
Q2 2012	69.3%	16.6%	3.1%	4.9%	6.1%

Figura 3.5.3: Porcentajes de uso de cada Sistema Operativo [13].

Cómo podemos observar en las dos anteriores estadísticas, Android es dueña del mercado con un 83% de dispositivos por lo que diseñar la aplicación nativamente para Android es un posible caso de éxito y llegará a más gente. Se valoró también utilizar otros *frameworks* mixtos, pero no dejaban trabajar al nivel requerido para el proyecto.

Capítulo 4

4 Desarrollo de la solución

Después de una leve introducción sobre las tecnologías que se utilizarán a lo largo de este proyecto, comenzamos por la solución a implementar en el Servidor-Robot.

Se ha utilizado un *desarrollo software iterativo e incremental (IID)*, hecho por una serie de iteraciones en las que se han implementado funcionalidades requeridas por nuestro software.

4.1 Servidor NodeJS "Servidor-Robot"

En nuestro caso se han elegido las siguientes dependencias declaradas en los *packages* del proyecto:

```
{
  "name": "ProjectRTC",
  "version": "0.3.2",
  "author": "Jesus Ramirez Ramos",
  "description": "Proyecto TFG UMA",
  "scripts": {
    "start": "forever start app.js",
    "stop": "forever stopall"
  },
  "dependencies": {
     "express": "4.13.3",
    "ejs": "2.3.4",
          "ftp": "0.3.10",
          "open": "*",
   "socket.io": "~0.9",
    "forever": "0.15.1",
          "mqtt":"1.5.0"
  }
}
```

Se procede a explicar porque se ha usado cada librería y qué propósito tiene cada una de ellas.

1. Express: es un *framework* de desarrollo de aplicaciones web minimalista y flexible para NodeJS. Es robusto, rápido, flexible y muy simple. Entre otras características, ofrece Router de URL (Peticiones *Get, Post, Put*), que son las que usaremos en nuestro proyecto.

2. Ejs: ES un lenguaje de plantillas *javascript* para NodeJS que fue originalmente parte de JavaScriptMVC y que permite crear estilos para interfaces web

fácilmente, en este caso se ha usado para crear la interfaz visual del servidor web, minimalista e intuitivo.

3. FTP: La elección de este módulo es fundamental en la conexión entre el cliente Android y el robot, puesto que esté ultimo al generar el token inicial con la contraseña se subirá a la red para que esté disponible para cualquier cliente Android para comodidad del usuario. Tal y como se puede observar en la figura 4.1.2 las credenciales para poder loguear al robot se depositan en el siguiente enlace: http://kasha-malaga.es/tfg.txt.

4. Open: Este módulo tiene un simple propósito, permitir las llamadas a la *shell* del sistema operativo en el que está corriendo NodeJS en nuestro caso, lo hemos usado para que al lanzar el servidor NodeJS este mismo, se encargue de llamar e iniciar el navegador predeterminado con la url a cargar.

5. Socket.io: Es un módulo que permite la comunicación socket, muy útil y usado en diferentes clientes de mensajería. En nuestra aplicación lo usamos en la gestión de sockets (manejadorSocket.js) para controlar la entrada/salida de peticiones; retransmitir los datos que nos llegan del cliente Android con la posición de movimiento x,y y grados de movimiento del cuello (Tilt) hacía el *broker* MQTT.

6. Foverer: Módulo muy útil, puesto que su funcionalidad es la siguiente; mantener un proceso hijo corriendo continuamente y automáticamente reiniciarse si ocurre un error inesperado.

   Si se produce una excepción que no está controlada, el servidor se quedaría parado, con dicho módulo tiene la habilidad de volver a lanzarse de forma automatizada, creándose un servidor perpetuo.

7. Mqtt: Es una biblioteca cliente para el protocolo MQTT, escrito en JavaScript. Con ella podremos enviar los mensajes a la pizarra de nuestro robot. Resumiendo, el uso de este módulo:

   Cuando se crea la conexión por primera vez:

   ```
   cliente  = mqtt.connect('mqtt://150.214.109.130'); //Conexión al servidor mqtt
   cliente.subscribe('UMA_Giraff_3/NavigationCommand'); // Nos suscribimos al
   cliente.emit('id', client.id); // le enviamos devuelta que ya estas conectado
   ```

   Cada vez que lleva un mensaje nuevo:

   ```
   cliente.publish('UMA_Giraff_3/NavigationCommand', options.name,0);
    // publico comando en mosquito con el topic Test //Motion X X
   ```

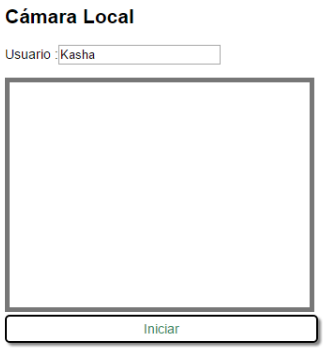


Figura 4.1.1: *Visión del pequeño servidor NodeJS en su primera fase [8].*

Lo importante de la interfaz web para el servidor es unicamente permitir el acceso hardware a los diferentes dispositivos de vídeo, por ello el resto de gestiones , así como información se muestran en la ventana de depuracion de NodeJS.

```
C:\Users\kasha\Documents\GitHub\UMA-ServidorWebRTC>node app.js
[Function]
Express server listening on port 3000
addr: 192.168.1.3
        -- Cliente con id:QlyKdqrEA71VzOzVPIbp conectado
        -- Creado servidor con la siguiente id: http://192.168.1.3:3000/QlyKdqrEA71VzOzVPIbp
Subiendo credenciales al FTP
```

Figura 4.1.2: *Visión de la imagen obtenida a través de la cámara web [9].*

4.2 Cliente WebRTC "Cliente Móvil Android"

Tal y como se ha comentado anteriormente en dicho proyecto se busca conseguir una interfaz estable y minimalista que permita la teleoperación de un robot de la forma más sencilla posible, permitiendo que cualquier usuario sin conocimiento pueda hacer uso del mismo.

Se ha intentado maximizar el uso de los dispositivos aceptados para la ejecución de dicha aplicación, siendo válidas las arquitecturas arm y x86.

También se ha propuesto como uso mínimo la API 15 de Android siendo posible su uso en dispositivos desde la versión 4.0.x

Primera iteración

En dicha iteración se montaron varios proyectos de prueba buscando cómo era más sencillo realizar la conexión entre el servidor NodeJS que estaba ya creado y el cliente; la librería *libjingle* así como los ejemplos del repositorio de Google eran bastante complejos y hacían uso de muchas funcionalidades extra que no nos eran útiles para el desarrollo. Por ello se fueron eliminando dichas funciones buscando tener una base estable sobre la que comenzar el desarrollo.

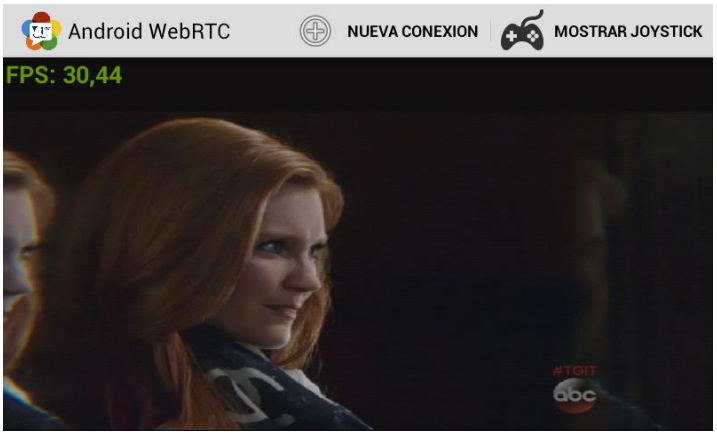


Figura 4.2.1: *15.7.2014 Audio y Video retransmitido al teléfono [10].*

En la imagen superior puede visualizarse un principio de lo que sería la base a tomar como inicio del proyecto; en este caso la imagen en vez de ser obtenida de una webcam, era un emulador de webcam que a su vez retransmitía una serie, la cual era reproducida en el dispositivo móvil. Comentar que en esta iteración la aplicación disponía de audio, por lo que era posible escuchar a través de los altavoces el audio proveniente del servidor.
Después de revisar lo conveniente para el proyecto se decidió dejar el canal de audio libre, pues WebRTC ofrece la posibilidad de deshabilitar el envío de audio y mejorar notablemente la imagen.

Aún no estaba planteado cómo diseñar los controles para dirigir al robot, pero se tenía una idea en mente, hacerlos de forma intuitiva y minimalista e intentar que no molestará para permitir la teleoperación fuese lo más sencilla posible.

Segunda iteración

La idea principal en esta iteración era dejar predefinidos como se diseñarían los controles, en este caso se optó por el control mediante joysticks, por su similitud al control remoto aplicado a los coches teledirigidos, después se comprobó si era factible pintar encima del *streaming* en tiempo real. Esto realmente fue uno de los retos más difíciles a conseguir, puesto que el componente *VideoStreamsView* pertenecía a la librería WebRTC y esta misma era una extensión de GLSurfaceView.

Después de numerosas pruebas, se consigue pintar en pantalla y mantener superpuesto por encima del video con una capa transparente unos círculos que imitaban la interfaz de un joystick, en dicho punto aún no se había decidido su diseño final.

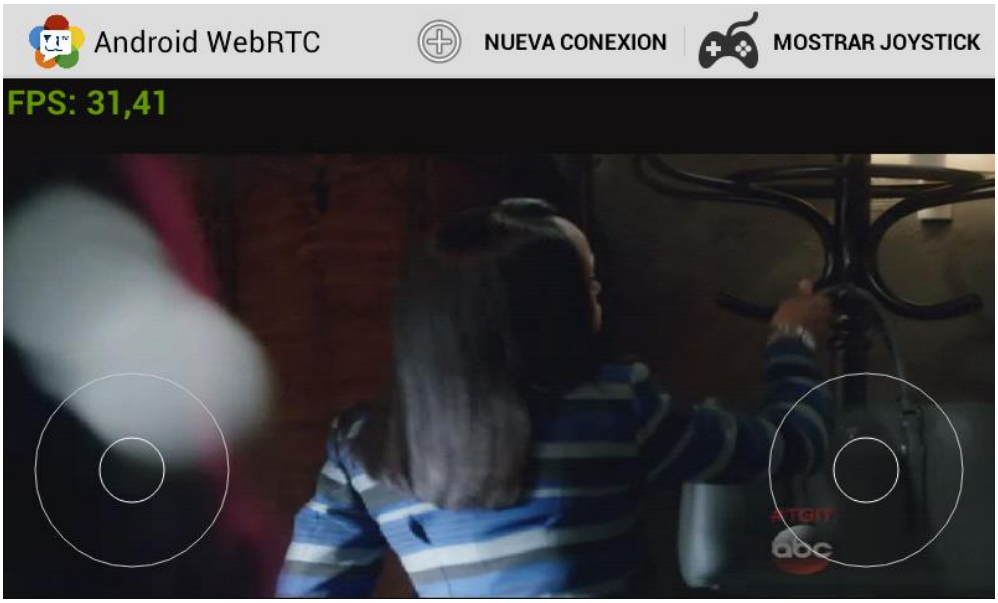


Figura 4.2.2: *16.7.2014 Círculos pintados encima del contenedor de video [11].*

Cómo se observa en la captura superior hemos decidido integrar dos joysticks, el izquierdo sería para el movimiento y el derecho para el movimiento del cuello, se barajaron diferentes opciones como usar una barra de desplazamiento para el cuello, pero finalmente se optó por dicha solución porque es la más cómoda a la hora de sujetar el terminal con ambas manos y deslizar los pulgares de forma intuitiva.

Tercera iteración

Una vez fue posible pintar encima y tener dos círculos pintados encima del video el siguiente paso fue pensar en cómo se podría reproducir el movimiento y la captura del mismo, por ello se buscaron diferentes ideas en internet sobre cómo podrían ayudarme las matemáticas a diseñar algo parecido a lo requerido para este proyecto. En este caso la idea se basaba en una opción visualizada en la web [5] http://www.zerokol.com/2012/03/joystickview-custom-android-view-to.html donde el movimiento del Joystick se basaba en dos parámetros: ángulo y potencia; de ellos era ya posible hallar la dirección.

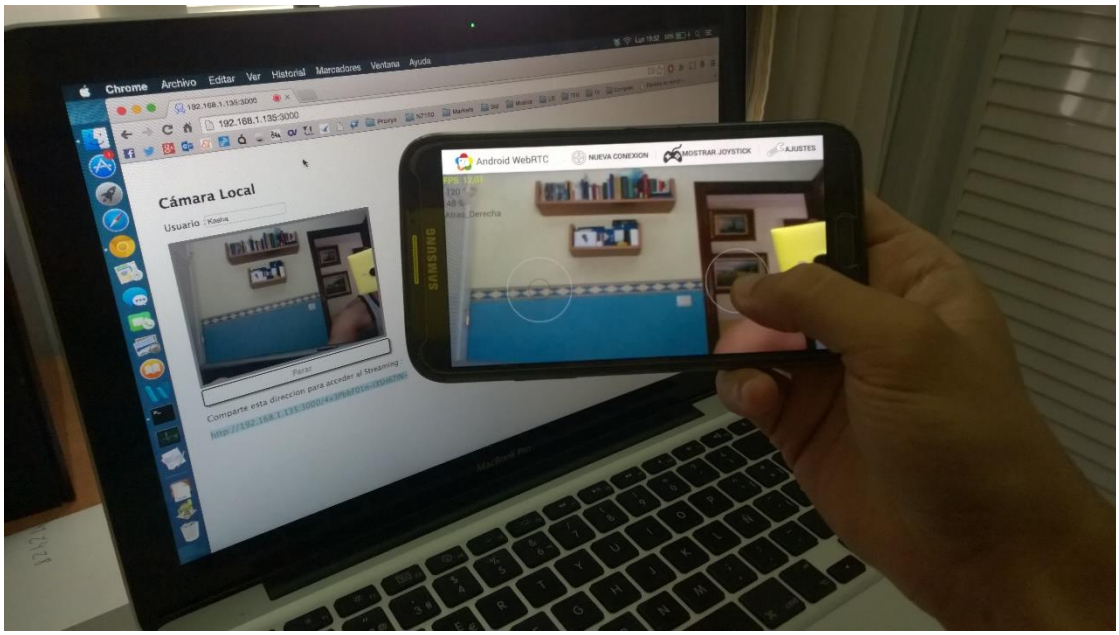


Figura 4.2.3: *4.08.2014 Joystick semi-funcional [12].*

Cuarta iteración

Con respecto a las necesidades necesarias a cubrir, se añaden unas opciones de ajustes, las cuáles anteriormente estaban una actividad aparte en otra capa superpuesta encima del vídeo, no siendo posible cambiar dichas preferencias en tiempo real y perdiendo el detalle de la vista ofrecida por dicho robot.

En dichas preferencias es posible mostrar más información técnica u ocultarla además de permitir cambiar los parámetros de velocidad, ángulo de giro del cuello, asistente al giro, todo ello en tiempo real.

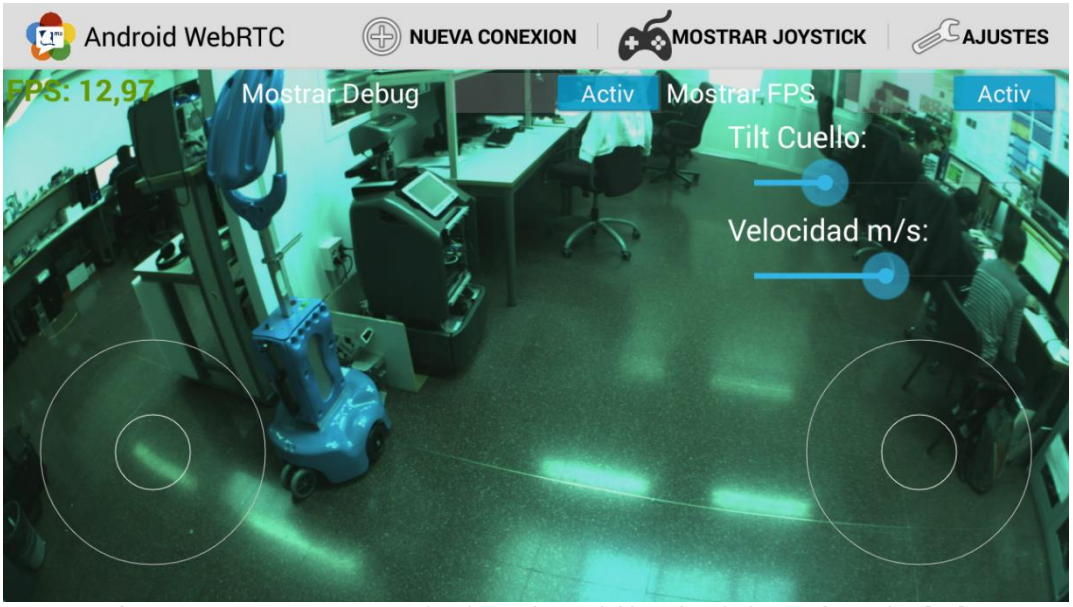


Figura 4.2.4: *Marzo 2015 Perfeccionamiento del interfaz de las preferencias [13].*

Cómo se aprecia en la imagen superior, ya está siendo probado el proyecto en un robot real para así conseguir depurar errores en los controles y dar comienzo a la fase de pruebas.

Última iteración

En esta última iteración se ha hecho un desarrollo en paralelo, mientras se realizaban pruebas de estabilidad en el robot, conforme se verificaba el funcionamiento y la estabilidad del mismo, se han pasado pruebas de cobertura y unidad mediante el *IDE* de Android Studio el cual ya dispone de dicho tipo de pruebas integradas.

En este caso se han elegido los parámetros óptimos para que el control fuese lo más acertado y preciso, se ha probado con un dispositivo móvil encerrado en una habitación, habilitar el control remoto, conectarse y realizar un circuito cerrado por los laboratorios de investigación.

Hemos dejado conectado el servidor NodeJS por numerosas horas para verificar la estabilidad del mismo, así como diversas pruebas, conexiones simultáneas desde diferentes dispositivos, etc.

4.3 Librerías externas

La aplicación Android se ha diseñado utilizando librerías nativas de Google aptas para el uso de WebRTC. Desde la versión Android 6, el componente WebView está integrado nativamente el protocolo WebRTC por lo que mediante un navegador es posible establecer dicha comunicación. Haciendo mucho más sencillo la implementación del componente.

Con el fin de ganar "cuota de mercado" se ha decidido desarrollar un cliente nativo… que permite la ejecución en dispositivos Android desde la versión 4 en adelante y, muy importante, con soporte arquitecturas de CPU ARM y x86, teniendo cubierto el porcentaje máximos de dispositivos del mercado.

Se ha tomado como inicio la información de los ejemplos que la propia Google pone a disposición de los desarrolladores en:
https://chromium.googlesource.com/external/webrtc/+/master/webrtc/examples/androidapp

Utilizando como base la aplicación de ejemplo proporcionada por Google se ha procedido a compilar las librerías de WebRTC (*Libjingle*) que deben ser compiladas en un proyecto suplementario.

Para más información extra sobre la organización de WebRTC puede visualizarse en el apéndice correspondiente a WebRTC (Cómo está organizado el protocolo y la pila de llamadas. [7]).

En los siguientes enlaces puedes encontrar referencias sobre como compilar o descargar las últimas versiones disponibles: [38] [39]
https://github.com/pristineio/webrtc-build-scripts
https://repo1.maven.org/maven2/io/pristine/libjingle_peerconnection_so/7113/

Dicha librería va integrarse en nuestro proyecto al igual que otras, siendo necesarias para el funcionamiento del proyecto.

```
dependencies {
    compile files('libs/libjingle_peerconnection-7113.jar')
    compile files("libs/lib_android_websockets.jar")
    compile files('libs/splunk-mint-4.3.0.jar')

}
```

Figura 4.3.1: Dependencias añadidas para el Cliente AndroidWebRTC [17].

En este caso se han añadido 3 dependencias externas:

- libjingle_peerconnection-7113.jar (Soporte Nativo WebRTC comentada más arriba)
- lib_android_websockets.jar (Permite crear conexiones *websocket* por la que se enviarán/recibirán los comandos de control del robot)
- splunk-mint-4.3.0.jar (Dicha librería se explica en la sección “Herramienta de depuración) [40].

4.4 Permisos en Android

Se va a comentar uno de los puntos más importantes de la aplicación, los permisos. Se ha buscado minimizar el uso de estos, eliminando los recomendados por WebRTC. Tal y como se muestra para un cliente audio-video en WebRTC en su ejemplo en: https://chromium.googlesource.com/external/webrtc/+/master/webrtc/examples/androidapp/AndroidManifest.xml

Se hacen uso de los siguientes permisos en la aplicación original de ejemplo:

```
<uses-permission android:name="android.permission.CHANGE_NETWORK_STATE"/>
<uses-permission android:name="android.permission.CAMERA" />
<uses-permission android:name="android.permission.MODIFY_AUDIO_SETTINGS"/>
<uses-permission android:name="android.permission.RECORD_AUDIO" />
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
```

Figura 4.4.1: *Permisos usados en la aplicación de demostración [18].*

Para el uso que hemos destinado en nuestro proyecto, buscando la minimización de permisos hemos conseguido reducir dichos permisos a lo que consideramos el mínimo para que nuestro proyecto funcione.

```
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
```

Figura 4.4.3: Permisos usados en la aplicación final [19].

Estos permisos son fundamentales e inamovibles, resumiendo su uso podemos decir que cada uno hace la siguiente función:

- Android.permision.INTERNET (Permite la interactividad del dispositivo con la conexión de internet.

- Android.permission.ACCESS_NETWORK_STATE (Le permite obtener el estado de la red (Wifi, 3G, 4G) y también comprobar si estamos conectados a una red.

4.5 Diagramas de clases de la aplicación Android

En este primer esquema, tenemos un diagrama de clases generalizado sin entrar en detalle sobre funciones, etc. Se va a explicar cómo se han diseñado y el uso de los métodos más importantes de cada clase que han sido necesarios para elaborar este proyecto.

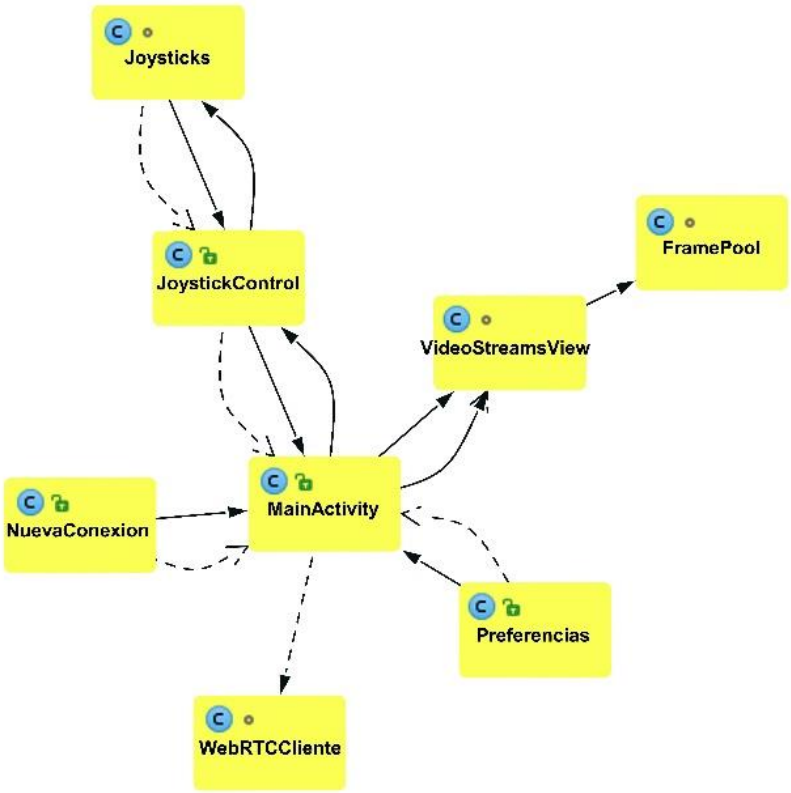


Figura 4.5.1: *Diagrama de clases Completo – Simplificado [14].*

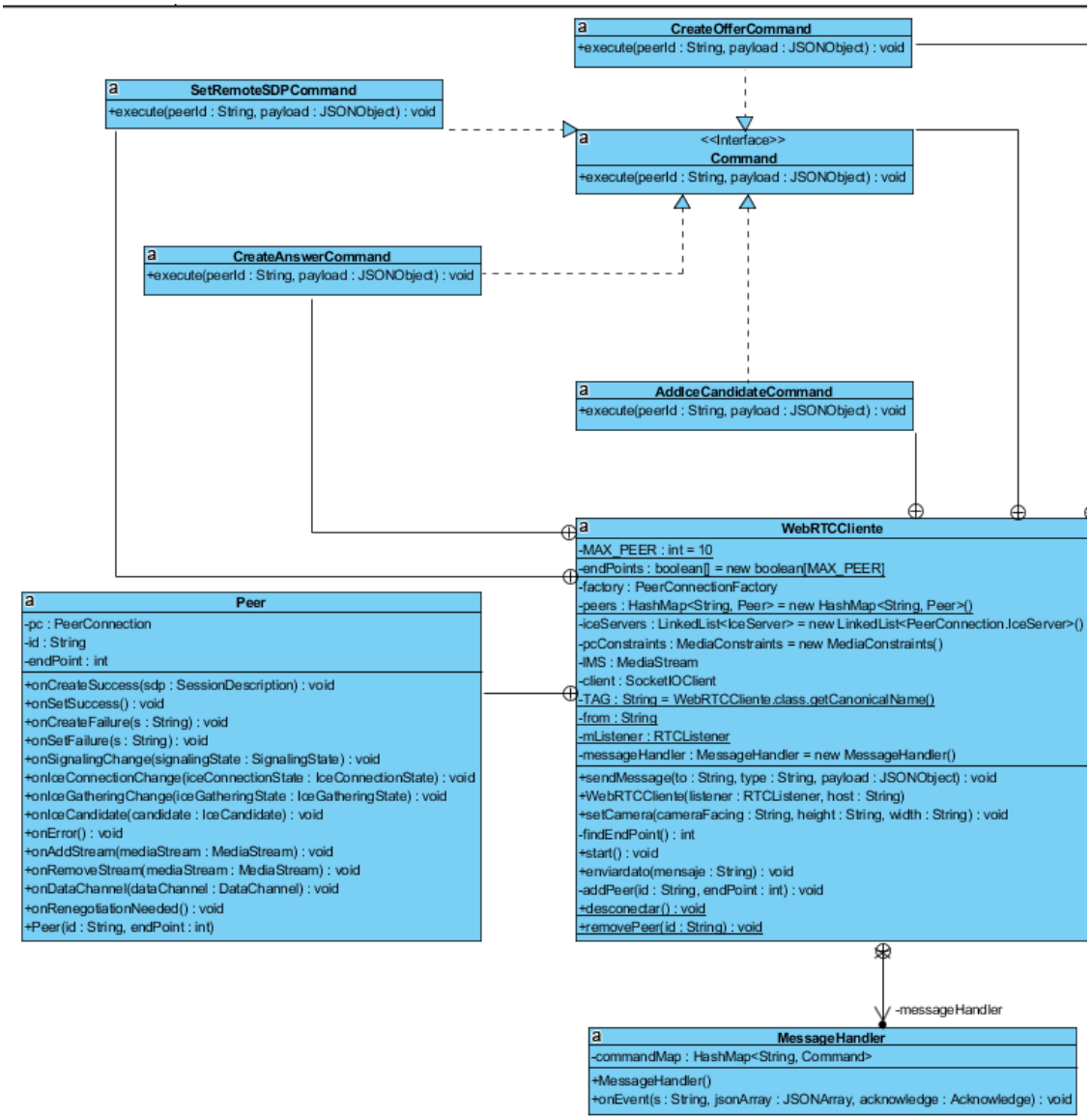


Figura 4.4.2: *Diagrama de Clases WebRTC [15].*

Vamos a enumerar las funciones más importantes de esta clase, la cual en parte es heredada del proyecto de Google, pero que ha tenido que retocarse para que encaje en el perfil de nuestro proyecto.

CreateOfferCommand: Crear una petición de ofrecimiento para el servidor WebRTC.

AddIceCandidateCommand: Es necesario añadir un cliente ICE que gestione la conexión.

WebRTCCliente
-MAX_PEER : int = 10: Establecimos un máximo de 10 clientes a la vez.
+WebRTCCliente(listener : WebRTCCliente$RTCListener, host : String):
El listener está a la espera para crear una conexión hacia un host determinado.

-peers : java.util.HashMap<String, es.kashamalaga.ClienteRTC.WebRTCCliente$Peer> = new HashMap<String, Peer>() Numero de compañeros disponibles.

+start() : void - Inicializa las peticiones pertinentes para generar el token de conexión.

+enviardato(mensaje : String) : void : Método usado para enviarDato a través del Websocket (en este caso usamos para movimiento y giro en grados del cuello).

-addPeer(id : String, endPoint : int) : void: Añade un nuevo cliente a la conexión
+desconectar() : void : Se cierra correctamente la conexión para que se pueda reanudar desde el mismo o desde otro dispositivo.

RTCListener
+onCallReady(callId : String) : void
+onStatusChanged(newStatus : String) : void
+onAddRemoteStream(remoteStream : org.webrtc.MediaStream, endPoint : int) : void
+onRemoveRemoteStream(remoteStream : org.webrtc.MediaStream, endPoint : int) : void

En este caso RTCListener es el manejador que controla el estado de la llamada por lo que de él depende de añadir o quitar video así como otros parámetros en la conexión.

Figura 4.4.3: *Diagrama de Clases MainActivity [16].*

Como se puede comprobar aquí está el grueso de la aplicación, aunque está modularizado, posee diferentes métodos y atributos.

-factoryStaticInitialized : boolean : Características WebRTC cargadas en memoria.

+CargarLibreriaNativaWebRTC() : void Función para inicializar y cargar en memoria la librería *libjingle*.

+cargarWeb2() : void Función que recupera de un servidor FTP la ruta de conexión hacía el servidor.

+NuevaConexion(mainActivity : es.kashamalaga.ClienteRTC.MainActivity)

Teniendo cargada una url hace el intento de establecer una nueva conexión.

+CargarPreferencias() : void
+GuardarPreferencias() : void

Gestiona y guarda los parámetros de la configuración para conservarlos en un fichero si la aplicación se cierra.

-joystickControl: es.kashamalaga.ClienteRTC.JoystickControl = new JoystickControl(this)

Aquí se une con la clase *Joystick* permitiendo crear la imagen de ambos cursores y su control.

<->vsvRenderizadoStream : es.kashamalaga.ClienteRTC.VideoStreamsView

Aquí se carga la clase *VideoStreamView* gracias a ella, se puede mostrar el video y se puede pintar encima los Joysticks.

+onCallReady(callId : String) : void
Cuando se han cargado todos los parámetros de conexión y se obtiene respuesta, dicha llamada permite que se inicialice la llamada.

+onBackPressed() : void
Método para controlar que ocurre cuando se pulsa el botón atrás en Android, en nuestro caso se controla la salida de la aplicación y guarda las preferencias.

+onCreateOptionsMenu(menu : android.view.Menu) : boolean
Crea el menú de la aplicación que permite realizar diferentes funciones (ocultar/mostrar Joystick, mostrar/ocultar preferencias, nueva conexión, desconectar).

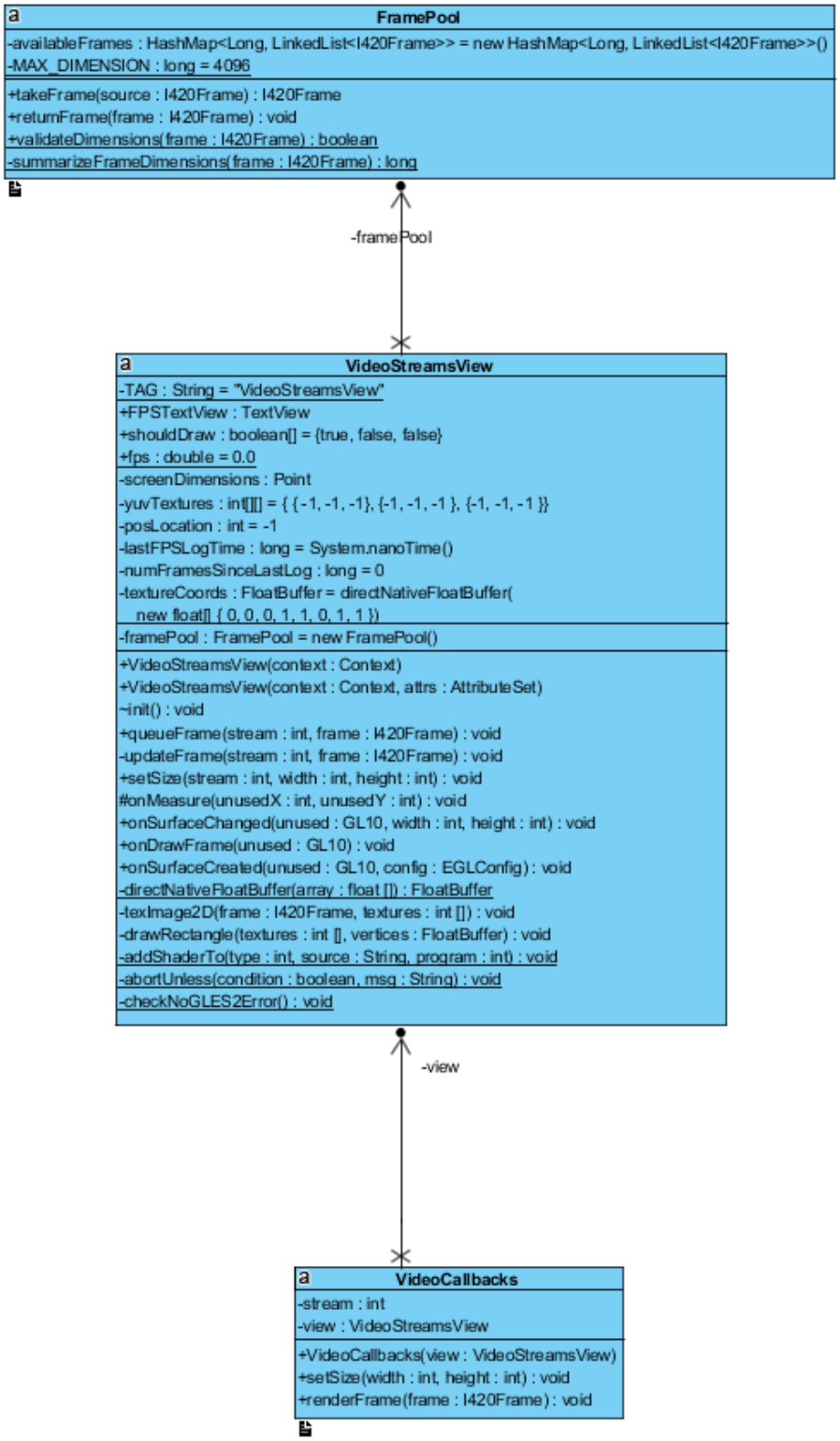


Figura 4.4.4: *Diagrama de Clases de VideoStreamView [17].*

-checkNoGLES2Error() : void

Función principal que permite conocer si el dispositivo permite hacer uso de las librerías OpenGL 2.0 sino es así, no sería posible usar la aplicación por una restricción de hardware.

+takeFrame(source : org.webrtc.VideoRenderer.I420Frame) : org.webrtc.VideoRenderer.I420Frame
Se toma una imagen de las que vienen desde el canal de video de WebRTC y se procesa.

-updateFrame(stream : int, frame : org.webrtc.VideoRenderer.I420Frame) :
Una vez procesado el frame se pide el siguiente, de forma ordenada, esto se usará para calcula el tiempo que transcurre entre uno y otro.

+renderFrameframe : VideoRenderer.I420Frame) : void
Una vez procesado el frame se envía y pinta en la pantalla gracias a esta función

-lastFPSLogTime : long = System.nanoTime()
Variable que toma el tiempo actual y que se usa para medir el tiempo transcurrido entre cada frame que entra, usándose como buffer para hacer los cálculos pertinentes y poder mostrar los frames por segundo al usuario. (Forma rudimental de comprobar la calidad de la conexión, 25fps constantes es lo ideal en una conexión aceptable).

-texImage2D(frame : org.webrtc.VideoRenderer.I420Frame, textures : int []) : void
Función que nos permite dibujar cualquier tipo de texto encima del componente *VideoStreamView.*

+setSize(width : int, height : int) : void
Establece unas dimensiones teniendo en cuenta el ancho y el alto del dispositivo, usado para generar un *VideoStreamView* con las medidas apropiadas.

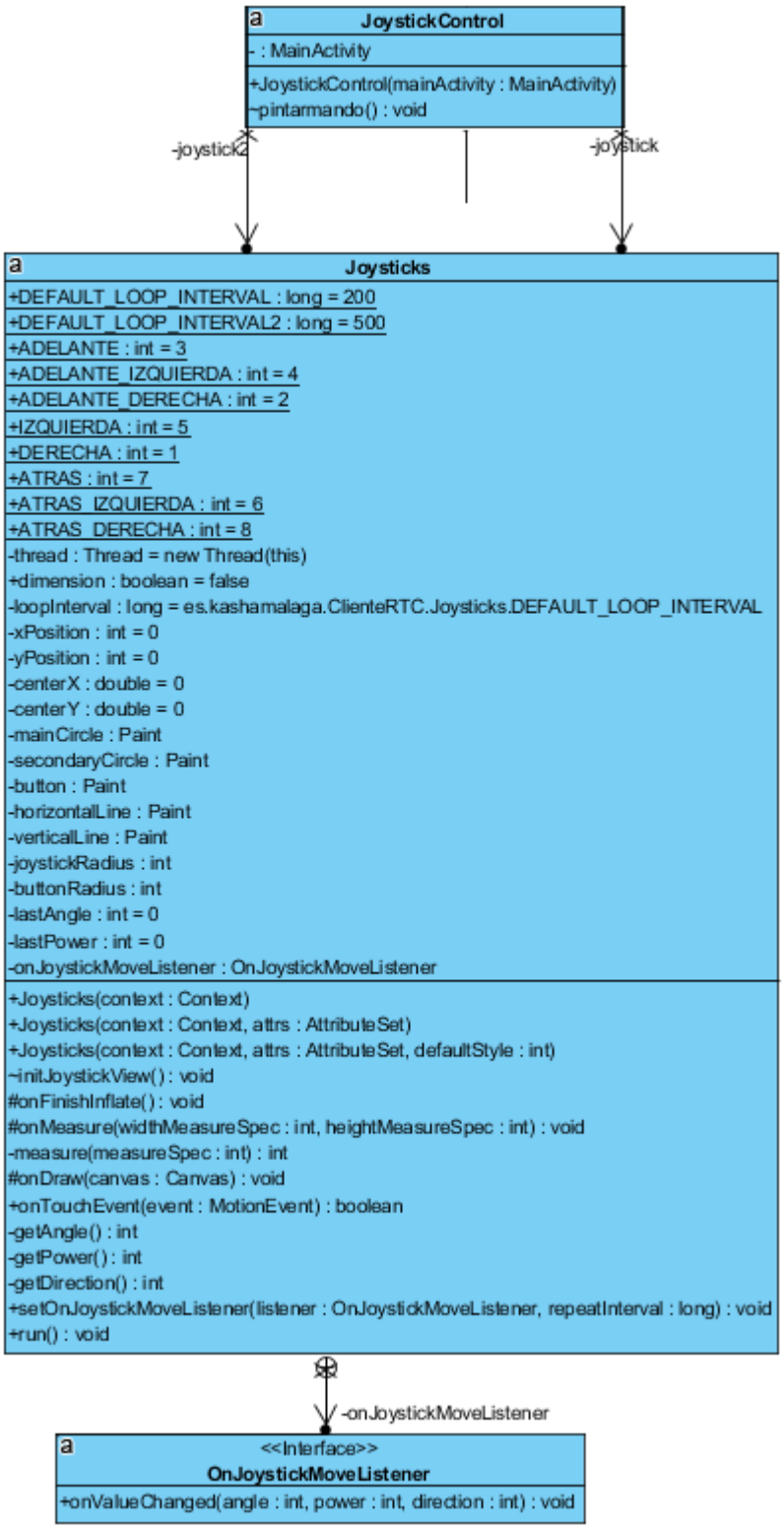


Figura 4.4.5: *Diagrama de Clases Joystick [18].*

~pintarmando() : void
Método usado para pintar los círculos que simulan ser los Joysticks.

-measure(measureSpec : int) : int
Método que indica las medidas necesarias para los círculos dependiendo del ratio asignado.

#onDraw(canvas : android.graphics.Canvas) : void
Método que nos permite dibujar encima de una capa ya pintada, en nuestro proyecto se usa para pintar fps, controles y ajustes.

+onTouchEvent(event : android.view.MotionEvent) : boolean
Método usado para controlar que ocurre al tocar en la zona del streaming, usado para mostrar/ocultar preferencias al doble toque.

-getAngle() : int
Nos devuelve el numero en grados, al tocar la pantalla dentro de la circunferencia.

-getPower() : int
Nos devuelve la potencia entre 0 y 100, para saber con qué velocidad y porcentaje debemos desplazarnos, usado tomando como 0% en el centro y 100% el extremo externo de la circunferencia del joystick.

-getDirection() : int
Según el número de grados devuelto por el método getAngle() podemos saber en qué dirección nos desplazamos, esto es utilizado solo como depuración para mostrarse por pantalla.

+setOnJoystickMoveListener(listener : es.kashamalaga.ClienteRTC.Joysticks$OnJoystickMoveListener, repeatInterval : long) : void
OnJoystickMoveListener
+onValueChanged(angle : int, power : int, direction : int) : void

Función que captura cada X ms representados por la variable repeatInterval los cambios que se producen al tocar el Joystick, seguido por lo que ocurre cuando cambia el valor del mismo.

Vamos a incluir un diagrama general para hacernos una idea del desarrollo así de como quedarían dichas clases unidas en su versión final.

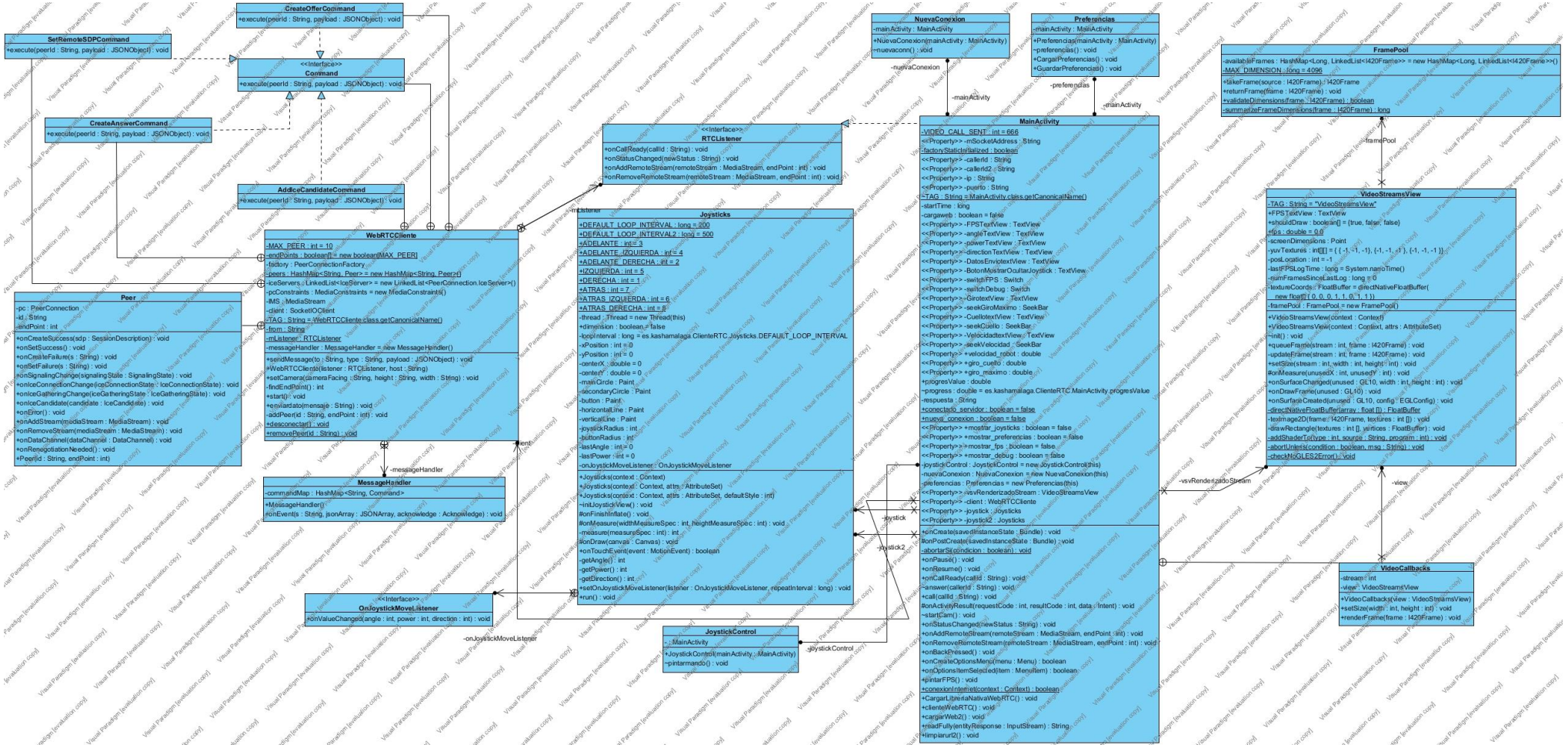

SetRemoteSDPCommand
+execute(peerId : String, payload : JSONObject) : void
CreateOfferCommand
+execute(peerId : String, payload : JSONObject) : void
<<Interface>>
Command
+execute(peerId : String, payload : JSONObject) : void
CreateAnswerCommand
+execute(peerId : String, payload : JSONObject) : void
AddIceCandidateCommand
+execute(peerId : String, payload : JSONObject) : void
Peer
-pc : PeerConnection
-id : String
-endPoint : int
+onCreateSuccess(sdp : SessionDescription) : void
+onSetSuccess() : void
+onCreateFailure(s : String) : void
+onSetFailure(s : String) : void
+onSignalingChange(signalingState : SignalingState) : void
+onIceConnectionChange(iceConnectionState : IceConnectionState) : void
+onIceGatheringChange(iceGatheringState : IceGatheringState) : void
+onIceCandidate(candidate : IceCandidate) : void
+onError() : void
+onAddStream(mediaStream : MediaStream) : void
+onRemoveStream(mediaStream : MediaStream) : void
+onDataChannel(dataChannel : DataChannel) : void
+onRenegotiationNeeded() : void
+Peer(id : String, endPoint : int)
WebRTCCliente
-MAX_PEER : int = 10
-endPoints : boolean[] = new boolean[MAX_PEER]
-factory : PeerConnectionFactory
-peers : HashMap<String, Peer> = new HashMap<String, Peer>()
-iceServers : LinkedList<IceServer> = new LinkedList<PeerConnection.IceServer>()
-pcConstraints : MediaConstraints = new MediaConstraints()
-lMS : MediaStream
-client : SocketIOClient
-TAG : String = WebRTCCliente.class.getCanonicalName()
-from : String
-mListener : RTCListener
-messageHandler : MessageHandler = new MessageHandler()
+sendMessage(to : String, type : String, payload : JSONObject) : void
#WebRTCCliente(listener : RTCListener, host : String)
+setCamera(cameraFacing : String, height : String, width : String) : void
-findEndPoint() : int
+start() : void
+enviardato(mensaje : String) : void
-addPeer(id : String, endPoint : int) : void
+desconectar() : void
+removePeer(id : String) : void
MessageHandler
-commandMap : HashMap<String, Command>
+MessageHandler()
#onEvent(s : String, jsonArray : JSONArray, acknowledge : Acknowledge) : void
-messageHandler
client
<<Interface>>
RTCListener
+onCallReady(callId : String) : void
+onStatusChanged(newStatus : String) : void
+onAddRemoteStream(remoteStream : MediaStream, endPoint : int) : void
+onRemoveRemoteStream(remoteStream : MediaStream, endPoint : int) : void
mListener
Joysticks
+DEFAULT_LOOP_INTERVAL : long = 200
+DEFAULT_LOOP_INTERVAL2 : long = 500
+ADELANTE : int = 3
+ADELANTE_IZQUIERDA : int = 4
+ADELANTE_DERECHA : int = 2
+IZQUIERDA : int = 5
+DERECHA : int = 1
+ATRAS : int = 7
+ATRAS_IZQUIERDA : int = 6
+ATRAS_DERECHA : int = 8
-thread : Thread = new Thread(this)
+dimension : boolean = false
-loopInterval : long = es.kashamalaga.ClienteRTC.Joysticks.DEFAULT_LOOP_INTERVAL
-xPosition : int = 0
-yPosition : int = 0
-centerX : double = 0
-centerY : double = 0
-mainCircle : Paint
-secondaryCircle : Paint
-button : Paint
-horizontalLine : Paint
-verticalLine : Paint
-joystickRadius : int
-buttonRadius : int
-lastAngle : int = 0
-lastPower : int = 0
-onJoystickMoveListener : OnJoystickMoveListener
+Joysticks(context : Context)
+Joysticks(context : Context, attrs : AttributeSet)
+Joysticks(context : Context, attrs : AttributeSet, defaultStyle : int)
-initJoystickView() : void
#onFinishInflate() : void
#onMeasure(widthMeasureSpec : int, heightMeasureSpec : int) : void
-measure(measureSpec : int) : int
#onDraw(canvas : Canvas) : void
+onTouchEvent(event : MotionEvent) : boolean
-getAngle() : int
-getPower() : int
-getDirection() : int
+setOnJoystickMoveListener(listener : OnJoystickMoveListener, repeatInterval : long) : void
+run() : void
<<Interface>>
OnJoystickMoveListener
+onValueChanged(angle : int, power : int, direction : int) : void
-onJoystickMoveListener
JoystickControl
- MainActivity
+JoystickControl(mainActivity : MainActivity)
-pintarmando() : void
-joystickControl
NuevaConexion
-mainActivity : MainActivity
+NuevaConexion(mainActivity : MainActivity)
-nuevaconn() : void
-nuevaConexion
Preferencias
-mainActivity : MainActivity
+Preferencias(mainActivity : MainActivity)
-preferencias() : void
+CargarPreferencias() : void
+GuardarPreferencias() : void
-preferencias
-mainActivity
-mainActivity
MainActivity
-VIDEO_CALL_SENT : int = 666
<<Property>> -mSocketAddress : String
-factoryStaticInitialized : boolean
<<Property>> -callerId : String
<<Property>> -callerId2 : String
<<Property>> -ip : String
<<Property>> -puerto : String
-TAG : String = MainActivity.class.getCanonicalName()
-startTime : long
-cargaweb : boolean = false
<<Property>> -FPSTextView : TextView
<<Property>> -angleTextView : TextView
<<Property>> -powerTextView : TextView
<<Property>> -directionTextView : TextView
<<Property>> -DatosEnviotextView : TextView
<<Property>> -BotonMostrarOcultarJoystick : TextView
<<Property>> -switchFPS : Switch
<<Property>> -switchDebug : Switch
<<Property>> -GirotextView : TextView
<<Property>> -seekGiroMaximo : SeekBar
<<Property>> -CuellotextView : TextView
<<Property>> -seekCuello : SeekBar
<<Property>> -VelocidadtextView : TextView
<<Property>> -seekVelocidad : SeekBar
<<Property>> +velocidad_robot : double
<<Property>> +giro_cuello : double
<<Property>> +giro_maximo : double
+progresValue : double
~progress : double = es.kashamalaga.ClienteRTC.MainActivity.progresValue
-respuesta : String
+conectado_servidor : boolean = false
+cuerpo_conexion : boolean = false
<<Property>> +mostrar_joysticks : boolean = false
<<Property>> +mostrar_preferencias : boolean = false
<<Property>> +mostrar_fps : boolean = false
<<Property>> +mostrar_debug : boolean = false
-joystickControl : JoystickControl = new JoystickControl(this)
-nuevaConexion : NuevaConexion = new NuevaConexion(this)
-preferencias : Preferencias = new Preferencias(this)
<<Property>> -vsvRenderizadoStream : VideoStreamsView
<<Property>> -client : WebRTCCliente
<<Property>> -joystick : Joysticks
<<Property>> -joystick2 : Joysticks
+onCreate(savedInstanceState : Bundle) : void
#onPostCreate(savedInstanceState : Bundle) : void
-abortarSi(condicion : boolean) : void
+onPause() : void
+onResume() : void
+onCallReady(callId : String) : void
-answer(callerId : String) : void
+call(callId : String) : void
#onActivityResult(requestCode : int, resultCode : int, data : Intent) : void
-startCam() : void
+onStatusChanged(newStatus : String) : void
+onAddRemoteStream(remoteStream : MediaStream, endPoint : int) : void
+onRemoveRemoteStream(remoteStream : MediaStream, endPoint : int) : void
+onBackPressed() : void
+onCreateOptionsMenu(menu : Menu) : boolean
+onOptionsItemSelected(item : MenuItem) : boolean
+pintarFPS() : void
+conexionInternet(context : Context) : boolean
+CargarLibreriaNativaWebRTC() : void
+clienteWebRTC() : void
+cargarWeb2() : void
+readFully(entityResponse : InputStream) : String
+limpiarurl2() : void
-joystick
-joystick2
FramePool
-availableFrames : HashMap<Long, LinkedList<I420Frame>> = new HashMap<Long, LinkedList<I420Frame>>()
-MAX_DIMENSION : long = 4096
+takeFrame(source : I420Frame) : I420Frame
+returnFrame(frame : I420Frame) : void
+validateDimensions(frame : I420Frame) : boolean
-summarizeFrameDimensions(frame : I420Frame) : long
-framePool
VideoStreamsView
-TAG : String = "VideoStreamsView"
+FPSTextView : TextView
+shouldDraw : boolean[] = {true, false, false}
+fps : double = 0.0
-screenDimensions : Point
-yuvTextures : int[][] = { { -1, -1, -1}, {-1, -1, -1}, {-1, -1, -1} }
-posLocation : int = -1
-lastFPSLogTime : long = System.nanoTime()
-numFramesSinceLastLog : long = 0
-textureCoords : FloatBuffer = directNativeFloatBuffer(
new float[] { 0, 0, 0, 1, 1, 0, 1, 1 })
-framePool : FramePool = new FramePool()
+VideoStreamsView(context : Context)
+VideoStreamsView(context : Context, attrs : AttributeSet)
-init() : void
+queueFrame(stream : int, frame : I420Frame) : void
-updateFrame(stream : int, frame : I420Frame) : void
+setSize(stream : int, width : int, height : int) : void
#onMeasure(unusedX : int, unusedY : int) : void
+onSurfaceChanged(unused : GL10, width : int, height : int) : void
+onDrawFrame(unused : GL10) : void
+onSurfaceCreated(unused : GL10, config : EGLConfig) : void
-directNativeFloatBuffer(array : float []) : FloatBuffer
-teximage2D(frame : I420Frame, textures : int []) : void
-drawRectangle(textures : int [], vertices : FloatBuffer) : void
-addShaderTo(type : int, source : String, program : int) : void
-abortUnless(condition : boolean, msg : String) : void
-checkNoGLES2Error() : void
-vsvRenderizadoStream
-framePool
-view
VideoCallbacks
-stream : int
-view : VideoStreamsView
+VideoCallbacks(view : VideoStreamsView)
+setSize(width : int, height : int) : void
+renderFrame(frame : I420Frame) : void

Figura 4.4.6: Diagrama de Clases Versión Final [19].

4.6 Diseño de la vista Cliente Android

Sobre la interfaz de la aplicación se muestran los elementos que la componen, así como su organización:

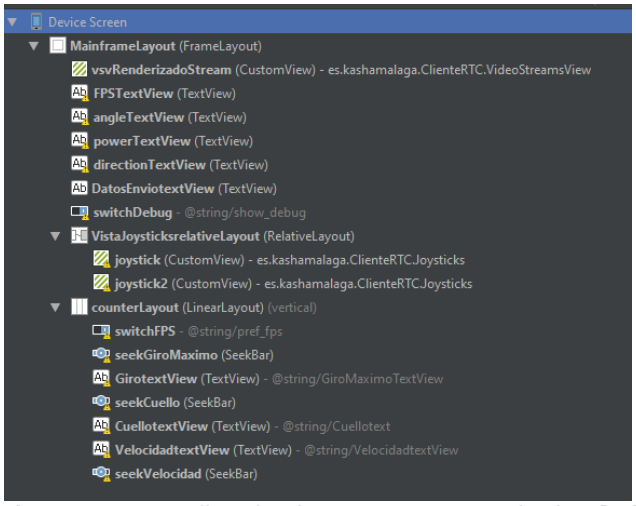


Figura 4.5.1: *Detalle sobre los componentes en la vista [20].*

Diseño XML generado para dichos componentes en su plantilla XML:

```
<?xml version="1.0" encoding="utf-8"?>
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/MainframeLayout"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:keepScreenOn="true"
    android:orientation="horizontal"
    android:background="@android:color/darker_gray" >


    <es.kashamalaga.ClienteRTC.VideoStreamsView
        android:id="@+id/vsvRenderizadoStream"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent">

    </es.kashamalaga.ClienteRTC.VideoStreamsView>


    <TextView
        android:orientation="horizontal"
        android:layout_width="100dp"
        android:layout_height="wrap_content"
        android:text=""
        android:id="@+id/FPSTextView"
        android:textColor="@android:color/holo_green_dark"
        android:textSize="18sp"
        android:visibility="invisible"
        android:layout_marginLeft="2dp" />

    <TextView
        android:orientation="horizontal"
```

```
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text=""
    android:id="@+id/angleTextView"
    android:textColor="@android:color/holo_green_dark"
    android:visibility="invisible"
    android:layout_marginTop="50dp"
    android:layout_marginLeft="2dp"
    android:background="@android:color/transparent" />

<TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text=""
    android:id="@+id/powerTextView"
    android:textColor="@android:color/holo_green_dark"
    android:visibility="invisible"
    android:layout_marginTop="@dimen/activity_separacion_texto_2"
    android:layout_marginLeft="2dp"
    android:background="@android:color/transparent" />

<TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text=""
    android:id="@+id/directionTextView"
    android:textColor="@android:color/holo_green_dark"
    android:visibility="invisible"
    android:layout_marginTop="70dp"
    android:layout_marginLeft="2dp"
    android:background="@android:color/transparent" />

<TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:textAppearance="?android:attr/textAppearanceMedium"
    android:id="@+id/DatosEnviotextView"
    android:textColor="@android:color/holo_red_dark"
    android:layout_gravity="bottom|center_horizontal"
    android:layout_marginBottom="5dp"
    android:visibility="invisible" />
<Switch
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/show_debug"
    android:id="@+id/switchDebug"
    android:layout_gravity="center_horizontal"
    android:showText="true"
    android:textColor="@android:color/white"
    android:visibility="invisible"
    android:layout_marginRight="50dp" />


<LinearLayout
    android:id="@+id/counterLayout"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="vertical"
    android:layout_gravity="center_horizontal|top">
```

```
<Switch
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/pref_fps"
    android:id="@+id/switchFPS"
    android:showText="true"
    android:layout_gravity="right"
    android:textColor="@android:color/white"
    android:visibility="invisible"
    android:layout_marginRight="5dp" />

<SeekBar
    android:layout_width="200dp"
    android:layout_height="wrap_content"
    android:id="@+id/seekGiroMaximo"
    android:layout_marginRight="10dp"
    android:layout_gravity="right|center_horizontal"
    android:max="2000"
    android:visibility="invisible"
    android:layout_marginTop="10dp" />

<TextView
    android:layout_width="300dp"
    android:layout_height="wrap_content"
    android:textAppearance="?android:attr/textAppearanceMedium"
    android:text="@string/GiroMaximoTextView"
    android:id="@+id/GirotextView"
    android:layout_gravity="center_horizontal|right"
    android:textColor="@android:color/white"
    android:layout_marginLeft="-120dp"
    android:layout_marginTop="-25dp"
    android:visibility="invisible"
    android:layout_marginRight="10dp" />

<SeekBar
    android:id="@+id/seekCuello"
    android:layout_width="200dp"
    android:layout_height="wrap_content"
    android:max="1000"
    android:indeterminate="false"
    android:layout_gravity="right"
    android:visibility="invisible"
    android:layout_marginRight="10dp"
    android:layout_marginTop="10dp" />

<TextView
    android:layout_width="300dp"
    android:layout_height="wrap_content"
    android:textAppearance="?android:attr/textAppearanceMedium"
    android:text="@string/Cuellotext"
    android:id="@+id/CuellotextView"
    android:textColor="@android:color/white"
    android:autoText="false"
    android:visibility="invisible"
    android:layout_marginRight="10dp"
    android:layout_gravity="center_horizontal|right"
    android:layout_marginTop="-25dp" />

<TextView
    android:layout_width="320dp"
    android:layout_height="wrap_content"
```

```
        android:textAppearance="?android:attr/textAppearanceMedium"
        android:text="@string/VelocidadtextView"
        android:id="@+id/VelocidadtextView"
        android:layout_gravity="center_horizontal|right"
        android:textColor="@android:color/white"
        android:visibility="invisible"
        android:layout_marginRight="10dp"
        android:layout_marginTop="20dp" />

    <SeekBar
        android:layout_width="200dp"
        android:layout_height="wrap_content"
        android:id="@+id/seekVelocidad"
        android:visibility="invisible"
        android:max="1000"
        android:layout_marginRight="10dp"
        android:layout_gravity="right"
        android:layout_marginTop="-30dp" />


    <RelativeLayout
        android:id="@+id/VistaJoysticksrelativeLayout"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:background="@android:color/transparent">

        <es.kashamalaga.ClienteRTC.Joysticks
            android:id="@+id/joystick"
            android:layout_height="wrap_content"
            android:layout_width="wrap_content"
            android:layout_alignParentLeft="true">
        </es.kashamalaga.ClienteRTC.Joysticks>

        <es.kashamalaga.ClienteRTC.Joysticks
            android:id="@+id/joystick2"
            android:layout_height="wrap_content"
            android:layout_width="wrap_content"
            android:layout_alignParentRight="true" >
        </es.kashamalaga.ClienteRTC.Joysticks>

    </RelativeLayout>

</LinearLayout>

</FrameLayout>
```

Figura 4.5.2: Desarrollo de la interfaz de la vista general en XML [29].

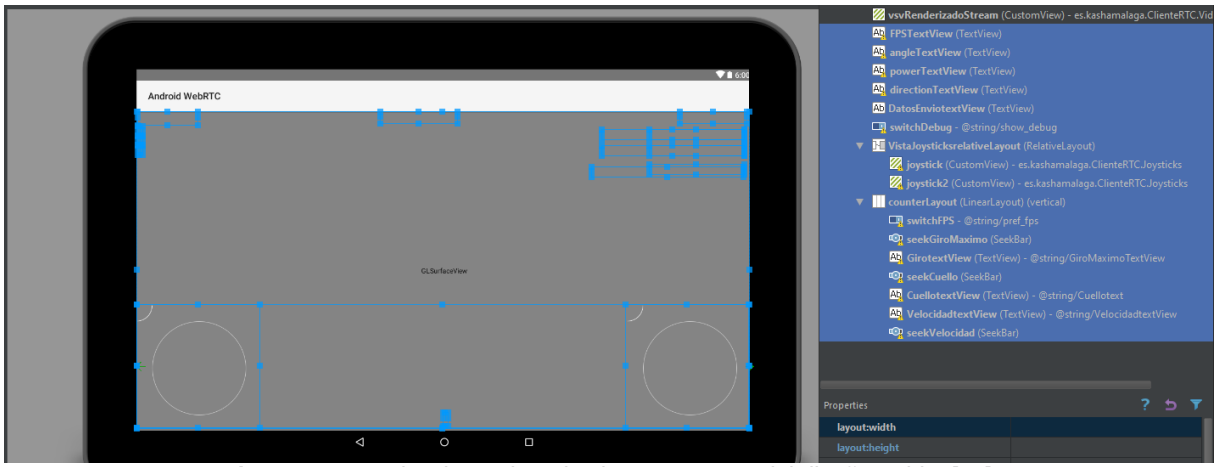


Figura 4.5.3: *Seleccionando todos los compontes del diseño Tablet [21].*

Cómo se puede observar algunos elementos de la inferfaz están ocultos por defecto y sólo se mostrarían si se activa una opción de depuración en la propia aplicación.

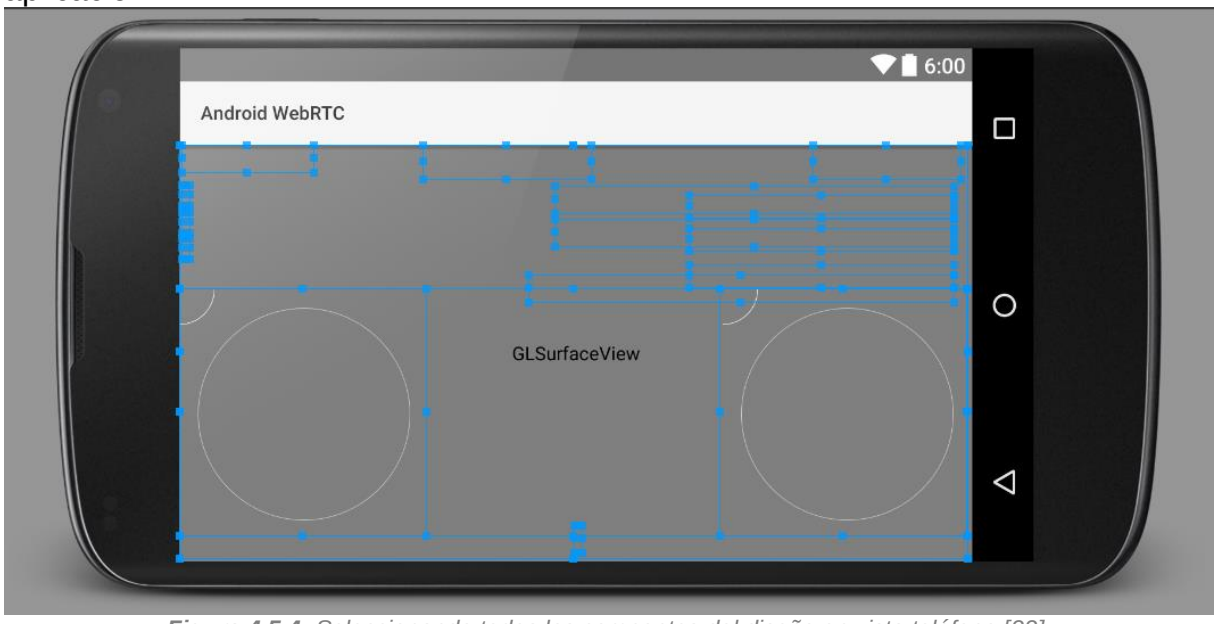


Figura 4.5.4: *Seleccionando todos los compontes del diseño en vista teléfono [22].*

Vamos ahora a mostrar cómo se ha diseñado el menú teniendo en cuenta que debe ocupar lo mínimo en la vista del usuario y que tiene funciones para adaptarse al tipo de tamaño en pantalla, así como la habilidad de esconderse al aplicar un doble toque a la pantalla.

```
<?xml version="1.0" encoding="utf-8"?>

<menu xmlns:android="http://schemas.android.com/apk/res/android">
    <item
        android:id="@+id/menu_nuevo"
        android:showAsAction="ifRoom|withText"
        android:icon="@android:drawable/ic_menu_add"
        android:title="@string/menu_nuevo"/>
    <item
        android:id="@+id/menu_habilitar_joystick"
```

```
        android:showAsAction="ifRoom|withText"
        android:icon="@drawable/joystick2"
        android:title="@string/mostrar_joystick"/>
    <item
        android:id="@+id/menu_preferencias"
        android:showAsAction="ifRoom|withText"
        android:icon="@android:drawable/ic_menu_preferences"
        android:title="@string/menu_preferencias"/>
    <item
        android:id="@+id/menu_desconectar"
        android:showAsAction="ifRoom|withText"
        android:icon="@android:drawable/ic_menu_close_clear_cancel"
        android:title="@string/menu_desconectar"/>
    <item
        android:id="@+id/menu_acercade"
        android:showAsAction="ifRoom|withText"
        android:icon="@android:drawable/ic_menu_info_details"
        android:title="@string/menu_acercade"/>

</menu>
```

Figura 4.5.6: Desarrollo de la interfaz del menú XML. [32].

***Figura 4.5.7**: Diseño de la vista del menú. [23].*

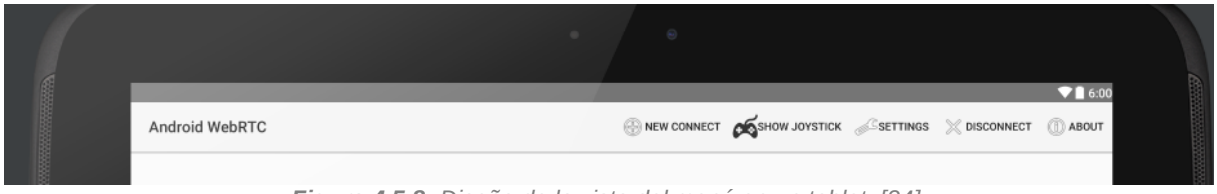


***Figura 4.5.8:** Diseño de la vista del menú en un tablet. [24].*

Capítulo 5

5 Pruebas de Rendimiento: SKYPE vs. WEBRTC

Han existido especulaciones acerca de WebRTC, augurando el fin de productos propietarios como *Skype* y *Facetime*. Siempre se nos ha referido a WebRTC como un "*Skype Killer*", pero es importante también tener en cuenta las diferencias entre ambos productos – WebRTC es una tecnología, no un producto.
Skype también se diferencia así mismo de WebRTC ofreciendo más que soporte de comunicaciones, pretendiendo así que sus clientes quedasen leales a su servicio. Un bloguero llamado *Rude Baguette* especuló, "el coste emocional de cambiar una solución por otra, suele ser mucho más alto que el coste económico".

Uno de los detalles a tenerse en cuenta: Microsoft, lejos de adoptar dicha nueva tecnología, ha decidido ponerse las pilas con Skype y orientar el servicio más a la Web al estilo de WebRTC, además de integrarlo en Windows, eliminando así la necesidad de los usuarios de tener que instalar complementos de terceros para realizar llamadas. Prefieras lo que prefieras, parece que por ahora existe un lugar para ambos (por ahora), pero podemos afirmar que, en un futuro muy cercano, se decantará por una solución única, y por ahora es WebRTC quien está por delante.
Durante una década, Skype ha sido una aplicación propietaria. Sin embargo, Microsoft ha conseguido, un gran número de usuarios y otras cosas más que hacen que aún siga siendo un peso pesado;

Si Microsoft y otras empresas hacen uso de las *URIs* (identificador de recursos uniforme, ejemplos: http: mailto: ftp:) para comunicaciones en tiempo Real, entonces Skype y WebRTC empezaran a parecerse más de cara a los usuarios.

WebRTC promete que el navegador será suficiente para utilizar todo lo que puede ofrecer dicha tecnología, pero aun, no tiene soporte de parte de Apple y Microsoft. Controvertidamente los clientes de Skype están disponibles para múltiples plataformas (Windows, OSX, iOS, Android, BlackBerry, Kindle, Smart TV´s, X-Box, PlayStation, etc. y además cuenta con las siguientes ventajas aquí resumidas frente a WebRTC:

- Gran número de usuarios hoy (299 millones de usuarios).
- Directorio de usuarios.
- Soporte Nativo PSTN (y SIP) sobre el signado de las comunicaciones.
- 100 millones de dispositivos móviles disponibles para usar Skype.
- Integración nativa con la plataforma para empresas (Lync).

WebRTC puede aún clamar que está en ventaja, al ser abierto, pero realmente nunca esto ha sido suficiente para ganar una apuesta en el terreno de las comunicaciones.

Quizás ni Microsoft/Skype puedan parar el avance de WebRTC. Es inevitable que el navegador soporte comunicaciones en tiempo real sin hacer uso de plugins de terceros. Sin embargo, Skype puede aún robar el momento de protagonismo, puesto que hay rumores; tienen pendiente presentar novedades en lo que a interfaces web se ofrece, pudiendo quizás Skype ofrecer sus servicios en la web, al más puro esto de la tecnología WebRTC, consiguiendo así iniciar llamadas con un solo click.

Para verificar efectivamente la superioridad del ancho de banda que se puede ahorrar haciendo uso de WebRTC, hemos hecho *una prueba en la que se conectan ambos clientes y realizan una conexión de 90 segundos, en las mismas condiciones, red*, etc. Así que a continuación se detallan los datos consumidos.

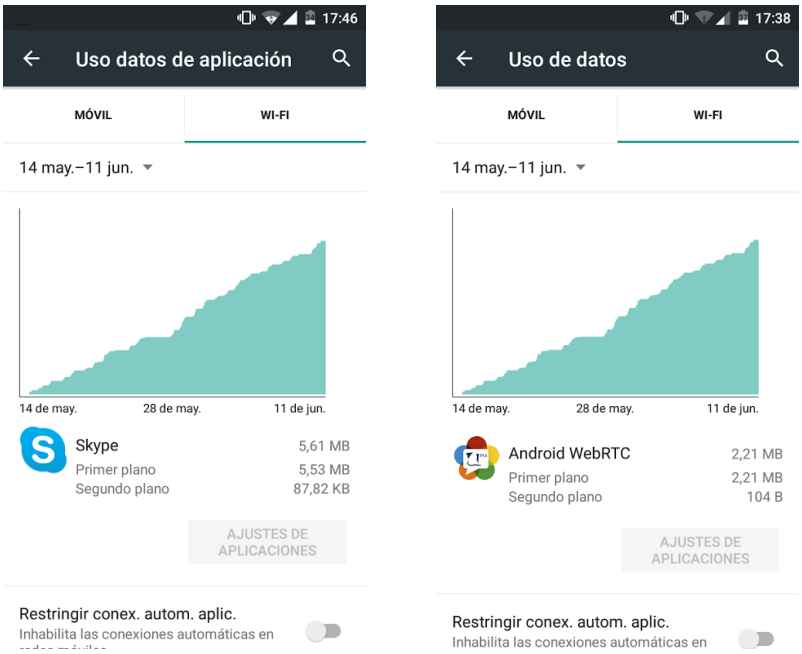


Figura 5.1.1: *Comparativa de ancho de banda WebRTC vs Skype [25]*

Cómo podemos comprobar el gasto de nuestra app Android WebRTC en 90 segundos son *2,21mb* mientras que la misma conexión establecida por Skype en 90 segundos son *5,61mb* y además la imagen fue más borrosa.

Se han realizado diversas pruebas en diferentes dispositivos y entornos siendo siempre WebRTC la que sale favorecida de dichos tests.

*Por norma general el rendimiento en WebRTC es de **un 60.61% mejor que Skype**.*

Por lo tanto, no dudamos que dichos datos se auguran muy interesantes teniendo en cuenta cómo avanzan las tecnologias móviles y cómo se ha de buscar el ahorro de ancho de banda en las tarifas móviles.
Hay rumores que dicen que Skype está trabajando en un protocolo mejorado basado en VP9 y otras fuentes apuntan a que próximamente se pasaran a WebRTC, sea cual sea la elección se avecinan cambios en las videoconferencias.

Capítulo 6

6 Conclusiones y futuras mejoras

6.1 Conclusiones finales

Tal y como se ha comentado en el apartado anterior de pruebas, se ha desarrollado una aplicación móvil capaz de teleoperar una plataforma robótica móvil mediante un protocolo de comunicaciones mucho más eficiente que lo utilizado anteriormente. WebRTC permite un ahorro efectivo del 61% en ancho de banda. Este ahorro hace posible desde cualquier sitio, teleoperar a los distintos robots que funcionan bajo la arquitectura robótica OpenMora de forma rápida e intuitiva.
En lo que respecta al control remoto se han barajado y experimentado con diferentes formas de control, siendo el joystick el control más intuitivo para los pulgares.

Trabajar con WebRTC cuando aún estaba en una fase tan temprana de desarrollo ha tenido una serie de dificultades, tanto la falta de ejemplos, como de documentación, además de errores que han requerido participar en los foros de desarrollo de Google.

En el momento en el que escribo estas líneas WebRTC ya es un estándar consolidado de la W3C con lo que su elección en su fase temprana fue todo un acierto.

Skype ha adoptado ya dicho protocolo adaptándolo a su manera con el nombre de ORTC en la web: https://www.w3.org/community/ortc/. Tal y cómo se comenta en diversos blogs (http://blog.thoughtstuff.co.uk/2015/05/skype-web-sdk-will-add-web-rtc-support-later-this-year/ desde el desembarco de Windows 10 en Julio de 2015, y su posterior actualización de Noviembre, ya integra soporte para ORTC en su navegador Microsoft Edge, siendo así posible realizar videoconferencias desde el propio navegador sin necesidad de instalar o requerir plugins de terceros.

Otras empresas como Facebook en su web de noticias: http://newsroom.fb.com/news/2015/04/introducing-video-calling-in-messenger/ también han adoptado dicha tecnología para sus videollamadas en su aplicación Messenger. Whatsapp por su parte también está migrando sus servicios desde XMPP hacía WebRTC con lo cual esto nos da una idea del cambio que ha supuesto dicha tecnología.

Por último, mencionar que, de los 4 objetivos que se fijaron al inicio de este proyecto, podemos afirmar que se han cumplido todos:

- Se han desarrollado con éxito los módulos software necesarios que permitan interconectar una aplicación Android con la arquitectura robótica utilizadas por los robots del grupo MAPIR.

- Conseguida una interfaz de usuario sencilla y ligera, que cumple los requerimientos establecidos.

- La solución que se ha propuesto consume muchísimos menos recursos y además es más rápida.

- La aplicación ha sido probada y validada por varios miembros del grupo de investigación MAPIR de la universidad de Málaga. También ha sido probada en personas de diferentes edades y rápidamente han conseguido entender el concepto y tener la teleoperación en sus manos.

6.2 Futuras líneas de trabajo

A continuación, se proponen diferentes líneas para continuar este proyecto, quedando así el camino abierto para añadir las siguientes funcionalidades:

- Recoger la actividad de sensores extras del robot, ya puede ser de cámaras *Kinect*, sensores de proximidad, láseres, etc.

- Crear un gestor de sesiones para gestionar mejor cuando se conectan diferentes personas a un mismo robot.

- Mejorar la interfaz de navegación, podrían incluirse mapas de navegación para entornos conocidos e inclusive recoger los datos de los generados durante la tarea de operación.

- Podría añadirse la posibilidad de enviar el audio del dispositivo móvil y que esté fuera reproducido en el robot y viceversa, en las versiones preliminares del sistema estaba implementado, por lo que no sería complicado el integrarlo nuevamente.

7 Referencias

[1] Altanai: "WebRTC Integrator's Guide". Packt Publishing Ltd, 2014, ISBN 978-17-8398-127-4 Versión Online.

[2] O'Reilly Media, Inc. "Real-Time Communication with WebRTC: Peer-to-Peer in the Browser", Septiembre de 2014, Salvatore Loreto Simón Pietro Romano ISBN 978-14-4937-183-8.

[3] Creando un cliente Android usando WebRTC [sitio web]. Disponible en: http://simonguest.com/2013/08/06/building-a-webrtc-client-for-android/. [Consulta: 1 de mayo de 2014].

[4] Proyecto WebRTC [sitio web]. Disponible en: https://code.google.com/p/webrtc/. [Consulta: 1 de mayo de 2014].

[5] Ejemplo práctico usando WebRTC [sitio web]. Disponible en: http://blog.davidvassallo.me/2014/02/07/got-15-minutes-easy-webrtc-android-app-in-3-steps/. [Consulta: 2 de mayo de 2014].

[6] Consejos básicos sobre el uso de WebRTC [sitio web]. Disponible en: http://ninjanetic.com/how-to-get-started-with-webrtc-and-ios-without-wasting-10-hours-of-your-life/. [Consulta: 5 de mayo de 2014].

[7] Ejemplo sobre un Joystick en Android [sitio web]. Disponible en: https://github.com/zerokol/JoystickView. [Consulta: 17 de mayo de 2014].

[8] Comparativa entre Skype - WebRTC [sitio web]. Disponible en: http://www.rudebaguette.com/2013/12/12/will-webrtc-defeat-skype-technology-does-not-a-product-make/. [Consulta: 27 de diciembre de 2013].

[9] Guía para creación de librerías nativas WebRTC [sitio web]. Disponible en: https://github.com/pristineio/webrtc-build-scripts. [Consulta: 17 de febrero de 2014].

[10] Repositorio con librerías ya compiladas de WebRTC [sitio web]. Disponible en: https://repo1.maven.org/maven2/io/pristine/libjingle_peerconnection_so/7113/. [Consulta: 18 de febrero de 2014].

[11] Guía básica sobre WebRTC [sitio web]. Disponible en: http://www.html5rocks.com/en/tutorials/webrtc/basics/#toc-rtcdatachannel. [Consulta: 15 de febrero de 2014].

[12] Funcionamiento sobre la API WebRTC - DataChannel [sitio web]. Disponible en: http://www.html5rocks.com/en/tutorials/webrtc/datachannels/. [Consulta: 15 de febrero de 2014].

[13] Dudas sobre el uso de Socket.io + Android con WebRTC [sitio web]. Disponible en: http://stackoverflow.com/questions/19674118/successfully-run-socket-io-android-project-with-nodejs. [Consulta: 1 de mayo de 2014].

[14] Cliente MQTT para NodeJS [sitio web]. Disponible en: https://github.com/mqttjs/MQTT.js. [Consulta: 30 de septiembre de 2014].

[15] Información sobre Cámaras Web WebRTC [sitio web]. Disponible en: https://es.wikipedia.org/wiki/C%C3%A1mara_web. [Consulta: 4 de mayo de 2014].

[16] Información sobre ordenadores personales WebRTC [sitio web]. Disponible en: https://es.wikipedia.org/wiki/Computadora_personal. [Consulta: 4 de mayo de 2014].

[17] Información sobre el internet de las cosas [sitio web]. Disponible en: https://es.wikipedia.org/wiki/Internet_de_las_cosas. [Consulta: 4 de mayo de 2014].

[18] Información sobre cómo aparecieron las cámaras web [sitio web]. Disponible en: https://en.wikipedia.org/wiki/Webcam. [Consulta: 4 de mayo de 2014].

[19] Información sobre los teléfonos que incorporaban videollamadas[sitio web]. Disponible en: https://en.wikipedia.org/wiki/Videophone. [Consulta: 4 de mayo de 2014].

[20] Información sobre el nacimiento de las conferencias vía internet [sitio web]. Disponible en: https://en.wikipedia.org/wiki/Videoconferencing. [Consulta: 1 de mayo de 2014].

[21] Información acerca del protocolo WebRTC [sitio web]. Disponible en: https://en.wikipedia.org/wiki/WebRTC. [Consulta: 15 de febrero de 2014].

[22] Ejemplos sobre WebRTC [sitio web]. Disponible en: http://thewebplatform.libsyn.com/web-rtc-and-designing-realtime-experiences. [Consulta: 17 de abril de 2014].

[23] Información sobre WebRTC frente a las comunicaciones VoIP [sitio web]. Disponible en: http://electronicdesign.com/communications/webrtc-and-vvoip-friends-or-enemies. [Consulta: 27 de mayo de 2014].

[24] Web sobre el proyecto WebRTC [sitio web]. Disponible en: https://bitbucket.org/webrtc/codelab. [Consulta: 17 de abril de 2014].

[25] Presentación sobre WebRTC [sitio web]. Disponible en: https://www.terena.org/activities/tf-webrtc/meeting1/slides/TF-WebRTC-Paris.pdf. [Consulta: 1 de mayo de 2014].

[26] Grupo de noticias sobre todo lo relacionado con WebRTC [sitio web]. Disponible en: http://www.webrtcworld.com/. [Consulta: 15 de diciembre de 2015].

[27] Comparativa sobre usos WebRTC vs VoIP [sitio web]. Disponible en: http://www.voip-info.org/wiki/view/WebRTC+vs+VoIP. [Consulta: 15 de octubre de 2015].

[28] Comparativa de cómo avanza WebRTC [sitio web]. Disponible en: http://blog.kundansingh.com/2014/12/webrtc-vs-sipsdp.html. [Consulta: 25 de febrero de 2015].

[29] Ligera introducción a WebRTC [sitio web]. Disponible en: https://www.onsip.com/blog/webrtc-sip-and-html5-brief-introduction. [Consulta: 14 de abril de 2015].

[30] Microsoft se replantea pasarse a WebRTC [sitio web]. Disponible en: http://venturebeat.com/2014/10/27/microsoft-eyes-webrtc-for-plugin-free-skype-calls-in-internet-explorer/. [Consulta: 15 de marzo de 2015].

[31] Información sobre NodeJS [sitio web]. Disponible en: https://es.wikipedia.org/wiki/Node.js. [Consulta: 20 de febrero de 2014].

[32] Empresas que usan NodeJS [sitio web]. Disponible en: https://nodejs.org/industry/. [Consulta: 20 de febrero de 2014].

[33] Guía de consejos sobre NodeJS [sitio web]. Disponible en: http://unadocenade.com/una-docena-de-conceptos-que-deberias-conocer-node-js/. [Consulta: 1 de mayo de 2014].

[34] Pequeña guía sobre OpenMora [sitio web]. Disponible en: http://openmora.github.io/#!tutorial-quickstart.md. [Consulta: 15 de mayo de 2014].

[35] WebRTC no se integrará con Skype, por ahora [sitio web]. Disponible en: http://www.webrtcworld.com/topics/webrtc-world/articles/365366-webrtc-will-not-push-skype-out-the-sandbox.htm. [Consulta: 15 de junio de 2014].

[36] Documento como WebRTC está compitiendo con Skype [sitio web]. Disponible en: http://www.nojitter.com/post/240158376/is-skype-a-webrtc-killer. [Consulta: 15 de julio de 2015].

[37] Información sobre Android [sitio web]. Disponible en: https://es.wikipedia.org/wiki/Android. [Consulta: 1 de mayo de 2014]

[38] Comparativa sobre el dominio de los sistemas operativos móviles [sitio web]. Disponible en: http://www.idc.com/prodserv/smartphone-os-market-share.jsp. [Consulta: 19 de octubre de 2015].

[39] Información sobre el robot Giraff [sitio web]. Disponible en: http://www.giraffplus.eu/. [Consulta: 18 de noviembre de 2015].

[40] Información sobre el robot Giraff [sitio web]. Disponible en: http://mapir.isa.uma.es/mapirwebsite/. [Consulta: 15 de enero de 2016].

[41] Información sobre compilación de librerías WebRTC [sitio web]. Disponible en: https://github.com/pristineio/webrtc-build-scripts. [Consulta: 26 de enero 2016].

[42] Información sobre compilación de librerías WebRTC [sitio web]. Disponible en:https://repo1.maven.org/maven2/io/pristine/libjingle_peerconnection_so/7113/. [Consulta: 26 de enero 2016].

Capítulo 8

8 Apéndices

8.1 Apéndice I: Código Fuente

Todo el código fuente desarrollado en este proyecto se encuentra disponible online en un repositorio GIT alojado en Github: https://github.com/KashaMalaga se pone este repositorio a disposición de los evaluadores para su consulta o sugerencias.

Además, se puede encontrar dicho código en el CD adjunto a este Trabajo Fin de Grado.

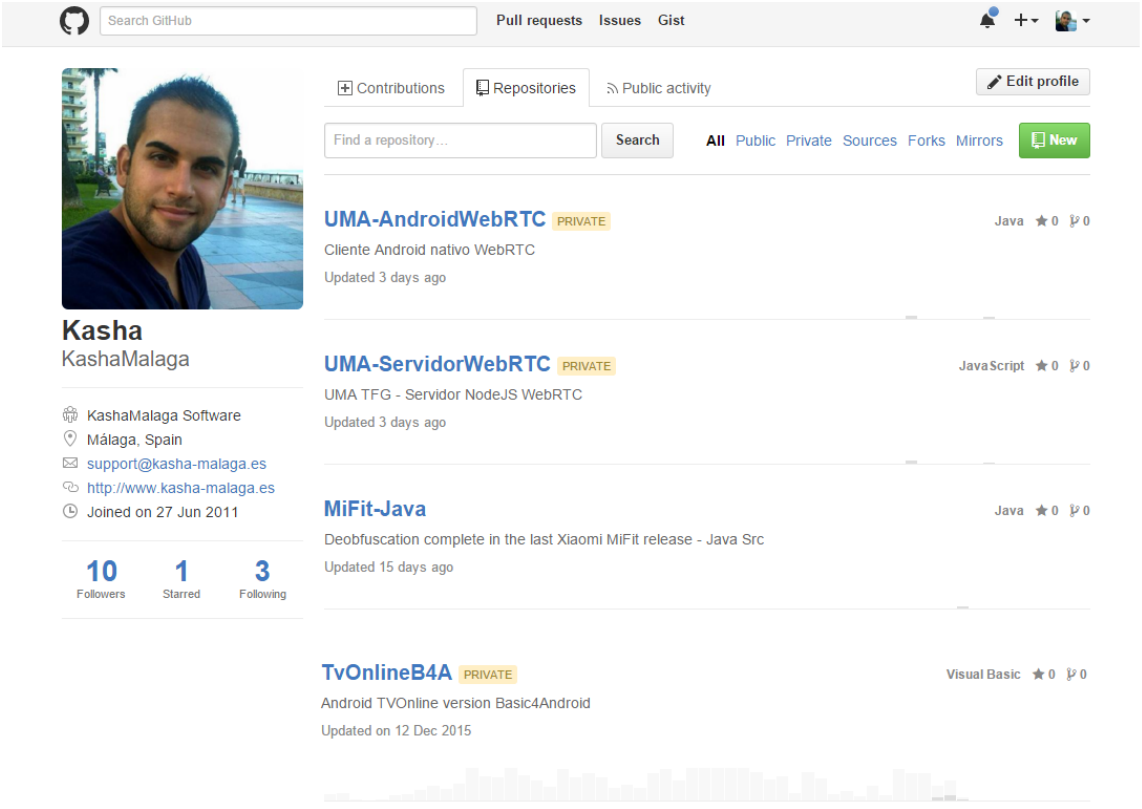


Figura 8.1.1: *Mi repositorio de Github con ambos proyectos [26].*

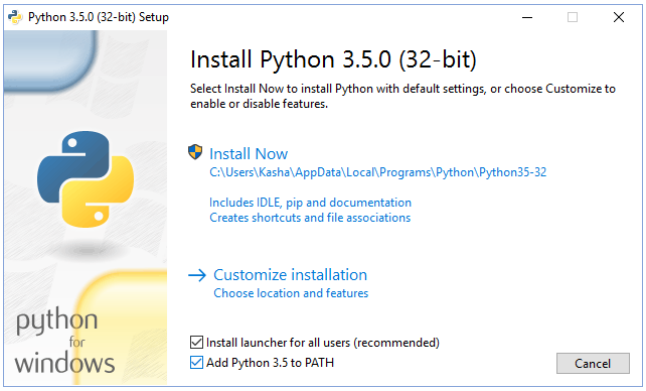
8.2 Apéndice II: Manual de usuario

Guía de instalación del servidor en un Robot Giraff:

Recordemos que nuestro equipo necesita la instalación de Python para poder compilar y ejecutar algunos módulos de nuestro servidor NodeJS.

Se procede a la instalación de **Python** en la url:
https://www.python.org/downloads/

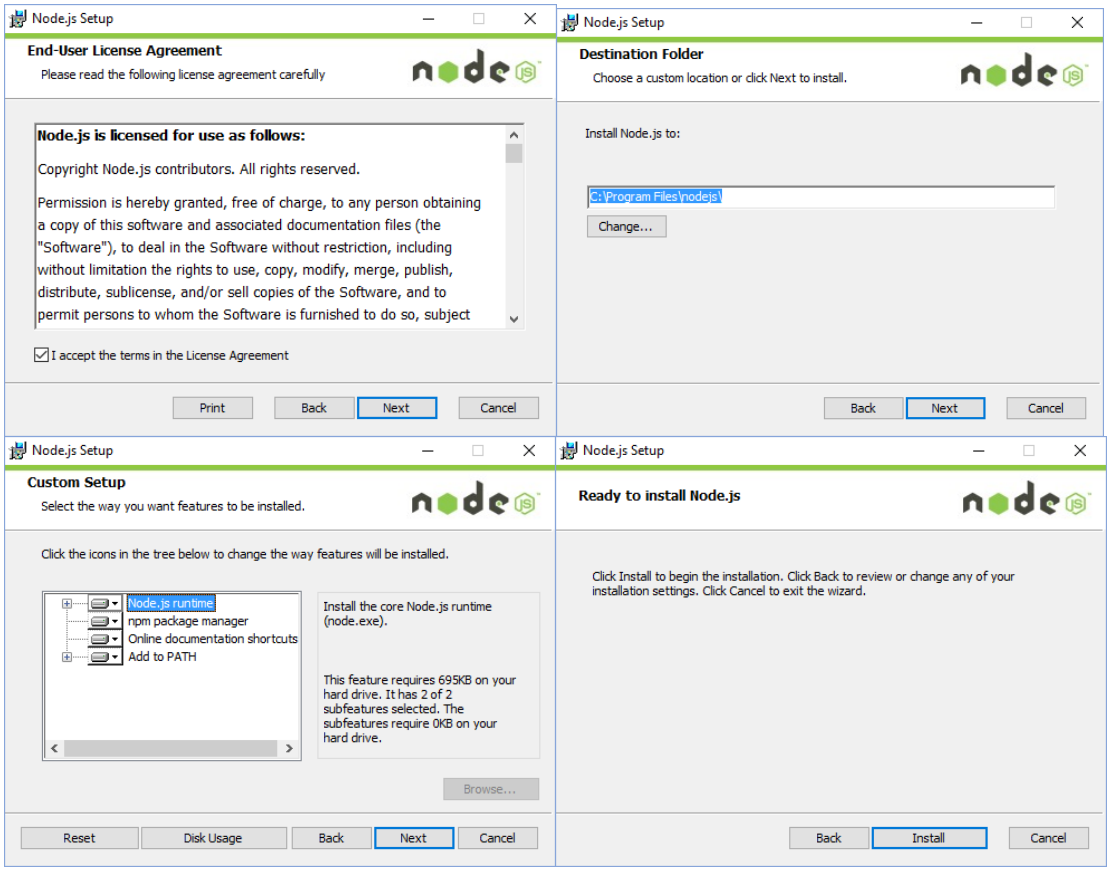
En nuestro caso se ha usado la versión 3.5.0
https://www.python.org/ftp/python/3.5.0/python-3.5.0.exe



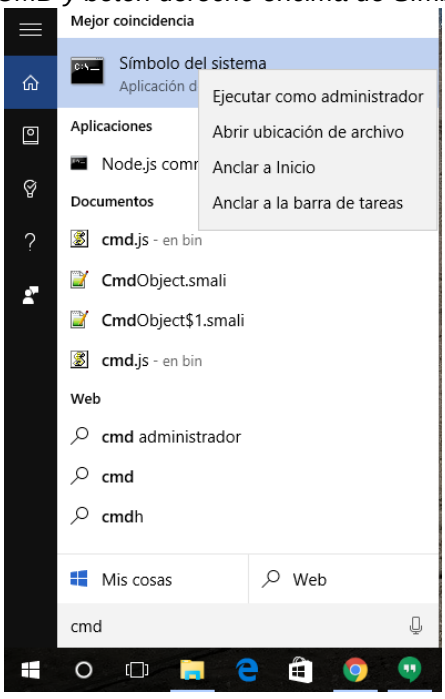
Ahora necesitamos instalar **NodeJS** visitaremos la web:
https://nodejs.org/en/

En la siguiente url se puede descargar el instalador (Versión 5.1.0):
https://nodejs.org/dist/v5.1.0/node-v5.1.0-x64.msi

Una vez descargado, comienza la instalación:

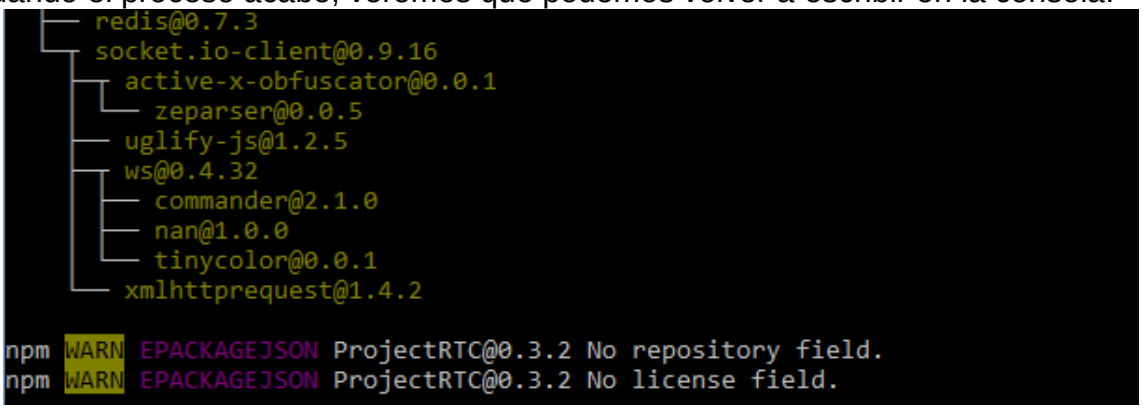


Una vez concluida la instalación, se procede a descargar el paquete **ServidorWebRTC** que contiene el proyecto NodeJS para poder instalar y compilar las dependencias necesarias para que funcione el proyecto.
Para ello necesitamos lanzar una ventana de comandos "con permisos de administrador". (*Escribiendo CMD y botón derecho encima de Símbolo del sistema*).

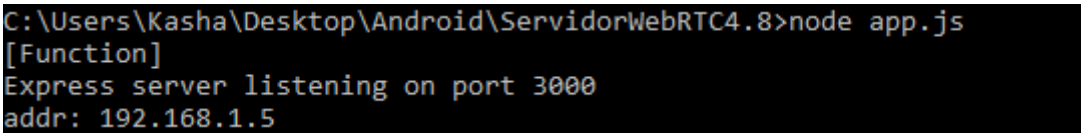


Ahora ya se puede ir a la ruta donde se ha descomprimido el proyecto ServidorWebRTC: *cd C:\Users\Kasha\Desktop\Android\ServidorWebRTC4.8*
Y ejecutaremos el comando "**npm install**".

Cuando el proceso acabe, veremos que podemos volver a escribir en la consola.



Ahora solo queda lanzar el servidor NodeJS haciendo uso del comando
"node app.js"



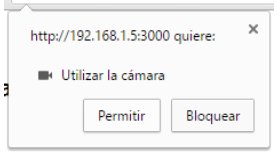
Como se puede ver en la captura superior ya ha sido lanzado el servidor y este está corriendo en la ip:192.168.1.5 .

Recordemos que el servidor funciona en el puerto 3000, por lo que será necesario usar la dirección del tipo: http://iplocalrobot:3000

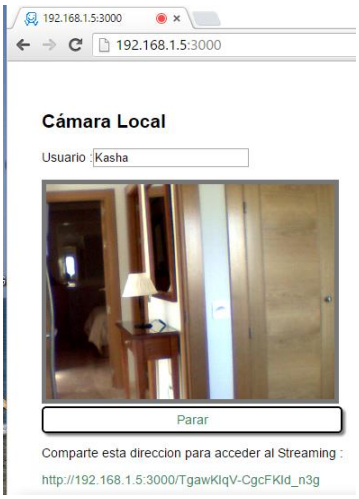
Es necesario usar un navegador compatible con el estándar WebRTC, en este caso hemos elegido Chrome 46.0.2490.86, no hemos usado versiones superiores porque desde noviembre de 2015, para el acceso hardware a una webcam exige de un servidor https con cifrado.

Se ha usado el navegador Chrome 46 disponible aquí:

Una vez que se ha conseguido iniciar nuestro navegador, en la url citada anteriormente pulsamos el botón Iniciar que nos pedirá el acceso al recurso de la cámara, por lo que ha de aceptarse



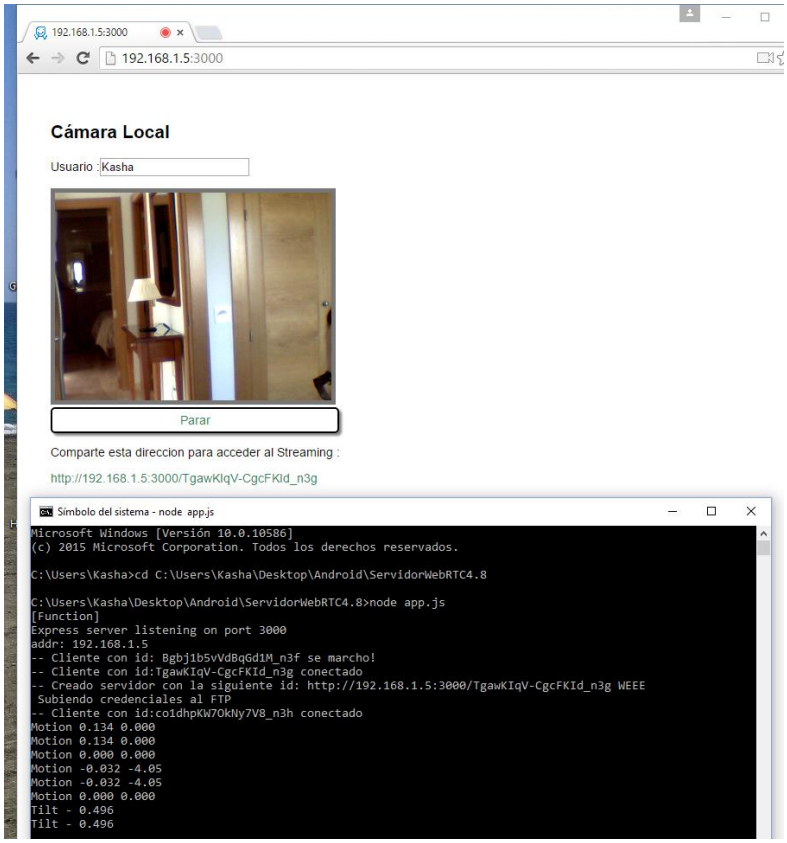
Si todo fue correctamente debería visualizarse el video de la cámara, así como la dirección de acceso generada.



Además, en el log debería aparece un texto tal que así:

```
-- Cliente con id:TgawKIqV-CgcFKId_n3g conectado
-- Creado servidor con la siguiente id: http://192.168.1.5:3000/TgawKIqV-CgcFKId_n3g WEEE
 Subiendo credenciales al FTP
```

Ahora que tenemos el servidor correctamente corriendo podemos olvidarnos de Chrome y por lo tanto quedar este minimizado, ya sólo nos quedaría ejecutar el cliente, que en este caso en una aplicación Android, para conseguir tener acceso a los recursos del robot.



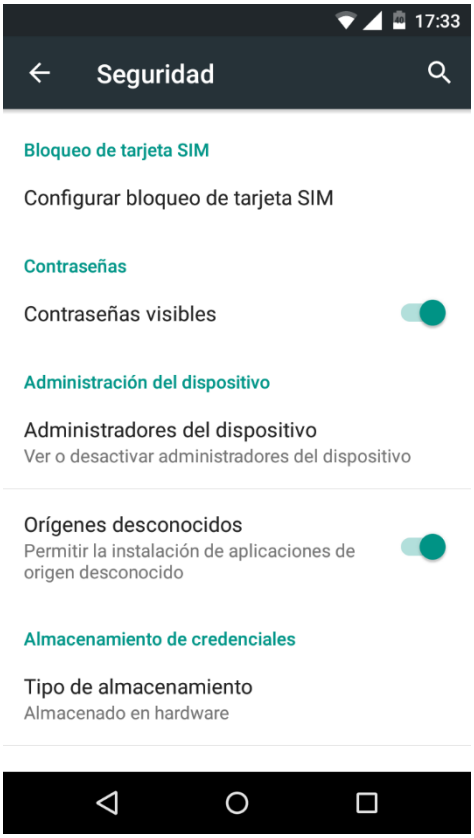
En la ventana del servidor NodeJS se aprecia como un cliente Android se ha conectado y está enviado comandos: *Motion (*x,y*)* y *Tilt* (giro del cuello en grados x).

Instalacion en la parte del Cliente Android:

Para empezar, necesitamos un dispositivo con Android 4 o superior, siendo válidas las arquitecturas de procesador ARM (usadas por casi todos los teléfonos/tabletas del mercado) e inclusive la arquitectura x86 usada por algunos dispositivos.

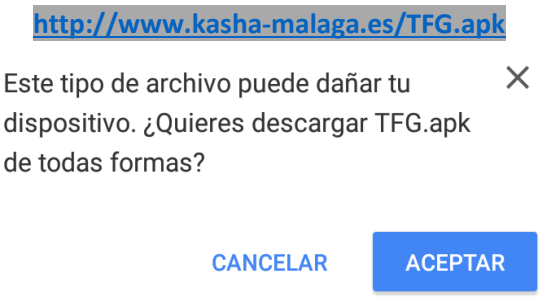
Antes de realizar la descarga de la aplicación se prepara el dispositivo para que pueda instalarse, sin problema alguno.

Para empezar, acudimos a Ajustes > Seguridad

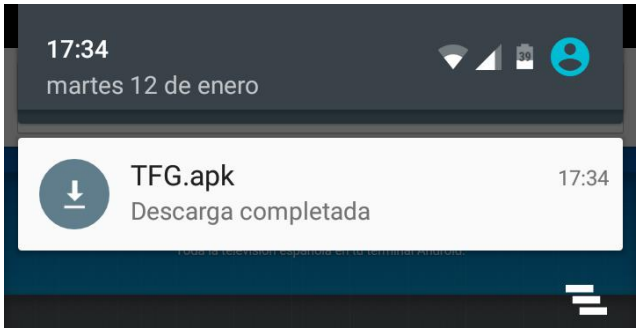


Tal y como nos muestra la imagen superior vamos activar la opción “Orígenes desconocidos” que nos permite instalar aplicaciones que no vienen de la tienda de Google Play; después podemos volver a desactivarlo por seguridad.

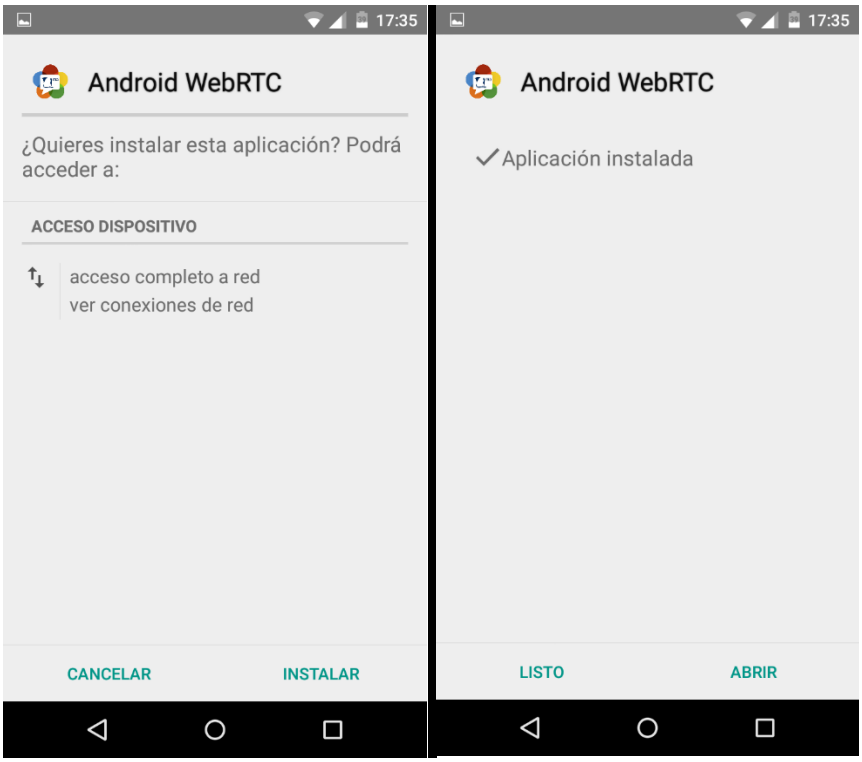
Para instalar el cliente debemos hacerlo desde la siguiente dirección (usando el Navegador Chrome Android en este caso) que estará habilitada temporalmente:



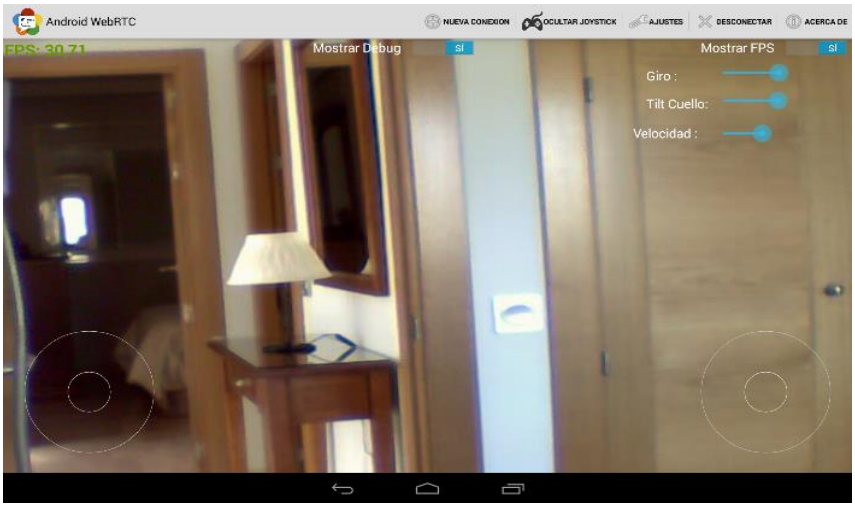
Pulsamos en aceptar para que la descarga se inicie, una vez completada, podemos iniciar la instalación desde el panel de notificaciones.



Como podemos ver los permisos usados para la aplicación, han sido minimizados lo máximo posible, quedando sólo el acceso de Internet requisito imprescindible.



Un ejemplo de la aplicación corriendo en distintos dispositivos (Teléfono y Tableta).



8.3 Apéndice III : Funcionamiento de WebRTC

En el siguiente apéndice vamos a intentar explicar cómo funciona el estándar de WebRTC comentando sus llamadas (APIs) principales, así como la exposición de un ejemplo básico (Alice y Bob).

MediaStream (aka *getUserMedia*):

MediaStream representa el flujo de datos multimedia obtenido de una cámara, también puede añadirse el audio de un micrófono. Cada sesión creada por la API de *MediaStream* tiene un identificador único, siendo este compartido con la otra persona para poder obtener dicha información. Dicha función viene en combinación del uso en *JavaScript* y otras librerías para obtener el acceso al hardware de la cámara.

Señalizaciones: sesiones de control, red e información multimedia.

WebRTC usa *RTCPeerConnection* para comunicar el flujo de datos entre dos navegadores (dos "peers" o compañeros), pero también necesita un mecanismo para coordinar la comunicación y enviar mensajes de control, un proceso conocido como señalización. Los métodos de señalización y protocolos no son especificados por WebRTC: La señalización no es parte de la API de *RTCPeerConnection*. En este caso, los desarrolladores de WebRTC pueden elegir cualquier protocolo de mensajería (normalmente SIP o XMPP) y hacer uso de comunicaciones en modo dúplex (dos vías).

En este caso la información que se enviar para la señalización es:

- Mensajes de sesión de control: para inicializar o cerrar la comunicación, así como reportar errores.
- Configuración de red: Fuera de casa cual es mi dirección pública y puerto para conectar a los equipos entre sí.
- Capacidad Multimedia: ¿Que códecs y resolución puede manejar mi navegador y puede ofrecer la cámara, puede el navegador soportarlo?

El intercambio de información vía señalización debe ser completado correctamente antes de iniciar la transmisión punto-punto (*peer* 2 *peer*).

Ejemplo básico de comunicaciones en WebRTC (Alice y Bob)

Imagina que Alice quiere comunicarse con Bob. En el siguiente fragmento se expone un ejemplo práctico de cómo se realizan las conexiones haciendo uso de WebRTC:

```
var signalingChannel = createSignalingChannel();
var pc;
var configuration = ...;
// lanzamos para inicializar la llamada
function start(isCaller) {
   pc = new RTCPeerConnection(configuration);
// Enviamos cualquier candidato hacia otro par
   pc.onicecandidate = function (evt) {
      signalingChannel.send(JSON.stringify({ "candidate": evt.candidate }));
   };
 //una vez el contenido remoto llega, lo mostramos en el contenedor de video
pc.onaddstream = function (evt) {
      remoteView.src = URL.createObjectURL(evt.stream);
   };
 // obtenemos el contenido local, mostrándolo en un contenedor de video y lo    //enviamos
navigator.getUserMedia({ "audio": true, "video": true }, function (stream) {
      selfView.src = URL.createObjectURL(stream);
      pc.addStream(stream);

      if (isCaller) // Si somos el que llama, creamos una oferta
         pc.createOffer(gotDescription);
      else // En caso de recibir una llamada, creamos la respuesta
         pc.createAnswer(pc.remoteDescription, gotDescription);

      function gotDescription(desc) {
         pc.setLocalDescription(desc);
         signalingChannel.send(JSON.stringify({ "sdp": desc }));
      }
   });
}
signalingChannel.onmessage = function (evt) {
   if (!pc)
      start(false);

   var signal = JSON.parse(evt.data);
   if (signal.sdp)
      pc.setRemoteDescription(new RTCSessionDescription(signal.sdp));
   else
      pc.addIceCandidate(new RTCIceCandidate(signal.candidate));
};
```

En el primer paso, Alice y Bob, intercambian información sobre su conexión. (La expresión encontrar candidatos se refiere al proceso de encontrar las tarjetas de red y puertos haciendo uso de “Framework ICE”.

1. Alice crea un objeto *RTCPeerConnection* con un manejador “*OnIceCandidate*”.
2. El manejador es lanzado en la red de candidatos, poniéndose disponible.
3. Alice envía los datos serializados a Bob, haciendo uso de la señalización elegida (por ejemplo, WebSocket como en nuestro proyecto).
4. Cuando Bob obtiene un mensaje candidato de Alice, él llama a la función “*addIceCandidate”*, para añadir el candidato a la descripción remota de conexión.

Los clientes WebRTC (conocidos como compañeros o “peers”, en nuestro caso Alice y Bob) también intercambian qué dispositivos tienen para intercambiar audio/video, así como la resolución de sus cámaras y su ancho de banda.

El intercambio sobre su configuración multimedia se realiza con el intercambio de una oferta/respuesta usando una sesión de descripción de Protocolo (SDP):

1. Alice lanza el método *createOffer*() en RTCPeerConnection. En el argumento que se envía; RTCSessionDescription, básicamente es la información sobre la sesión de Alice, así como su información multimedia, (códecs y resolución).
2. En la llamada de vuelta, Alice facilita la información descriptiva haciendo uso del método *setLocaDescription*() y entonces envía esa sesión (SDP) a Bob.
3. Bob también tiene que enviar a Alice su información haciendo uso del método *setRemoteDescription*().
4. Bob lanza entonces el método *createAnswer*(), pasando la sesión remota que obtuvo de Alice para que pueda comprobarse si es compatible con ella.
5. Cuando Alice obtiene la descripción de la sesión creada por Bob, ella también responde con su descripción remota para confirmar así la conexión.

   En la traza de datos siguiente puede verse, como es internamente un objeto RTCSessionDescription, (SDP).

```
v=0
o=- 3883943731 1 IN IP4 127.0.0.1
s=
t=0 0
a=group:BUNDLE audio video
m=audio 1 RTP/SAVPF 103 104 0 8 106 105 13 126

// ...

a=ssrc:2223794119 label:H4fjnMzxy3dPIgQ7HxuCTLb4wLLLeRHnFxh810
```

   Este intercambio de información sobre los dispositivos de red e información multimedia debe ser realizado simultáneamente, pero ambos procesos deben completarse antes, para poder iniciarse el vídeo.

   La oferta/respuesta descrita como JSEP (JavaScript Session Establishment Protocol), se muestra en la imagen inferior.

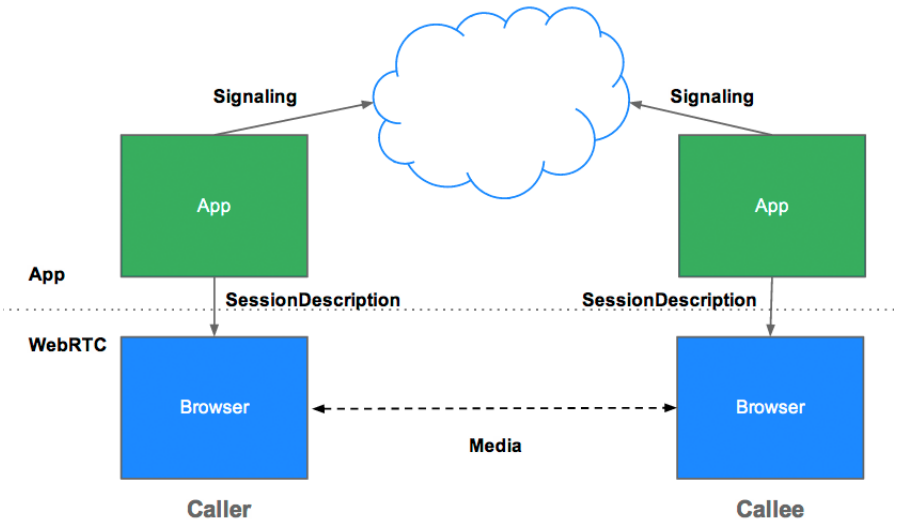


Figura 8.3.1: *Cliente –Servidor WebRTC [27].*

Una vez la señalización ha concluido con éxito, ya estamos listos para que comience el tráfico de datos con audio/video entre par y par, entre el que llama [*Caller*] y el llamado [*Callee*]; o si existe fallo, mediante un intermediario (servidor de retardo) ya que así podremos recuperar la sesión sin necesidad de generar un nuevo token.

Ahora vamos a estudiar la parte del tráfico de datos, que pertenece a RTCPeerConnection.

RTCPeerConnection:

En este caso, RTCPeerConnection es el componente de WebRTC que maneja la comunicación de manera eficiente y estable, permitiendo el intercambio de audio/video mediante los pares.

Más abajo hay una imagen que explica cómo funciona la arquitectura de WebRTC mostrándonos el verdadero rol de RTCPeerConnection. Como se puede observar en la imagen, las tareas más complejas son las verdes que son resueltas por la capa WebRTC en el propio navegador.

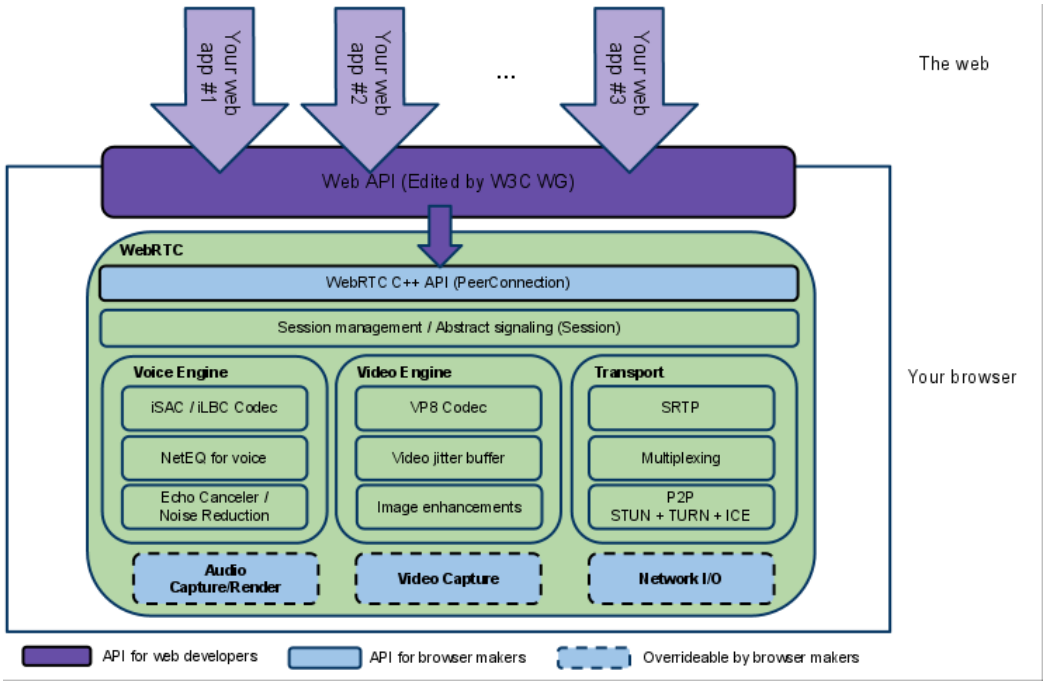


Figura 8.3.2: *Arquitectura WebRTC [28].*

Si tenemos en cuenta lo que realmente se usaba hasta hace relativamente poco tiempo, teníamos el paradigma del cliente-servidor en la web, donde el navegador envía una petición http al servidor y este, respondía con el contenido solicitado.

Los recursos servidos por el servidor iban asociados a una entidad conocida como *URI* (*Uniform Resource Identifier*) o URL (*Uniform Resource Locator*). En las aplicaciones web, el servidor podía ya contener algún código JS en html para ejecutar algún tipo de función en el navegador. Este tipo de interacción ha cambiado radicalmente al uso de las nuevas APIs y de las interfaces de usuario.

Nuestro proyecto hace uso de la tecnología WebRTC que, aunque semánticamente extiende de una conexión cliente-servidor, ha introducido un gran cambio, nadie es cliente y nadie es servidor***, ambos practican ambos roles a la vez***.

Desde una perspectiva JavaScript, lo que debe entenderse desde este diagrama es que RTCPeerConnection debe proteger a los desarrolladores Web de todas las complejidades y problemas internos.

Los códecs y protocolos usados por WebRTC realizan diversas operaciones para permitir la conexión en tiempo real, incluso sobre redes que no tienen un buen ancho de banda, teniendo integradas funciones como las que se citan:

- Ocultación de paquetes perdidos
- Cancelación de eco/ruidos
- Ancho de banda variable
- Buffer dinámico
- Control automático de ganancia
- Reducción de ruidos en las imágenes, así como compresión
- “Limpieza” de imágenes.

El uso de RTCPeerConnection, fuera de una red local, requiere el uso de otras herramientas para garantizar la conectividad:

- Usuario descubre a otro, obteniendo detalles como su nombre
- Un cliente WebRTC (par) intercambia información sobre su conexión.
- Los pares intercambian datos sobre su conectividad e información multimedia (códecs, formato de video, resolución).
- Cliente WebRTC traspasa en este caso los firewalls y routers.

En otras palabras, WebRTC necesita cuatro tipos de funcionalidades en un servidor para funcionar:

- Conectividad para permitir comunicación y descubrimiento de usuarios.
- Manejador de señales.
- NAT/ firewall transversal.
- Servidores de retardo en caso de que la comunicación punto a punto falle.

Los routers que permiten la conexiones punto a punto y los requerimientos para montar una aplicación servidor, nos ayudan a descubrir a usuarios y gestionar las señales, también se mencionaran en este proyecto.

Debemos comentar que STUN es un protocolo y extensión de TURN y es usado por el ICE Framework para permitir al componente RTCPeerConnection atravesar los firewalls y otras protecciones de redes existentes. En este caso el Framework ICE usado para conectar pares, tales como el enviado de video entre clientes. Inicialmente ICE intenta conectar directamente a los pares con la menor latencia posible vía UDP. En este proceso los servidores STUN tiene una única tarea: permitir a un par detrás de un router, encontrar su dirección pública y puerto. (Google en este caso tiene unos cuantos servidores de STUN, en este caso nosotros en nuestro proyecto se usa de uno de ellos).

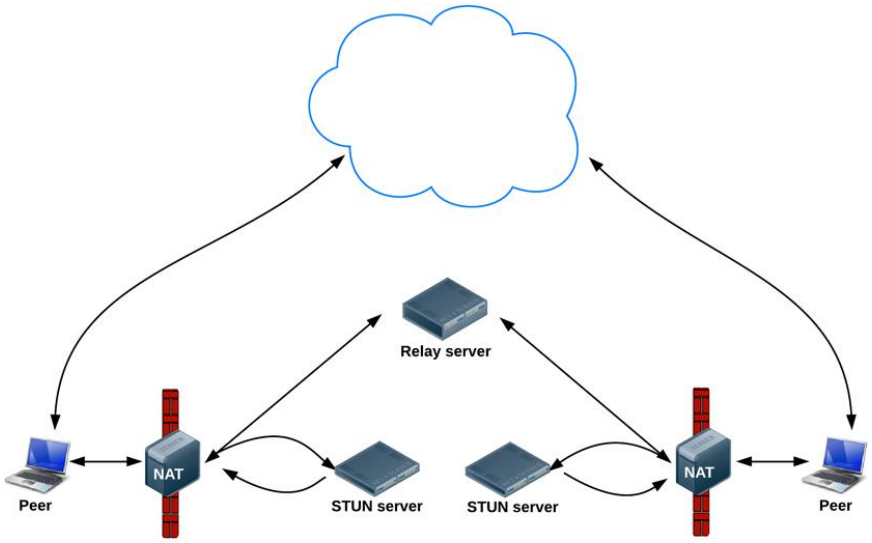


Figura 8.3.3: *Encontrando candidatos (proceso de encontrar interfaces de red y puertos) [29].*

Si UDP falla, ICE intenta entonces hacer uso de TCP: primero HTTP y después HTTPS, aunque existen planes para que sólo se permitan conexiones seguras en poco tiempo.

Si directamente la conexión falla – en particular, porque un router de empresa o firewall interviene en la conexión – ICE usa un servidor intermedio (*relay* TURN *server* o retardo como hemos comentado arriba).

En otras palabras, ICE primero intenta hacer uso de STUN por UDP, si falla, pasa a TCP.

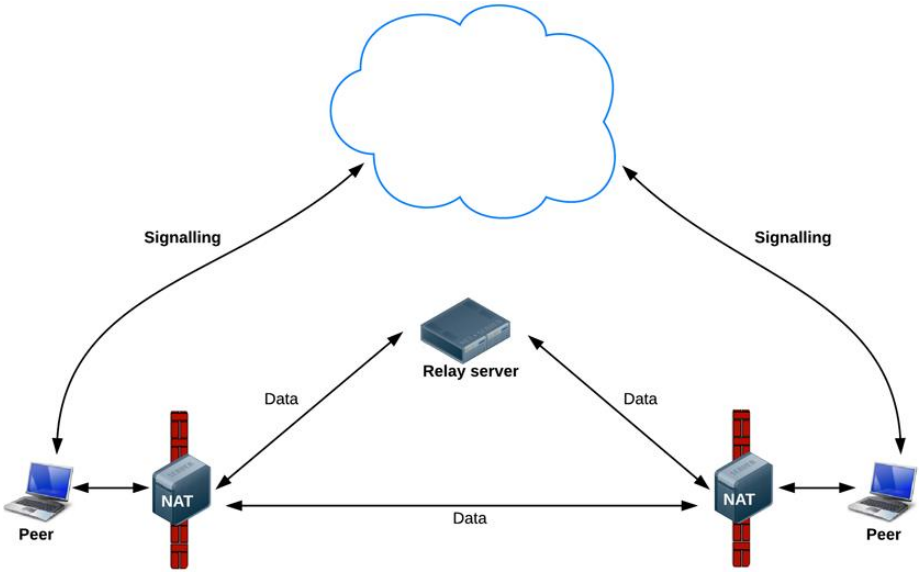


Figura 8.3.4: *Flujo de datos en WebRTC [30].*

Un buen lugar para comenzar a estudiar esta nueva tecnología, es tomar como ejemplo la aplicación de videochat de Google llamada appRTC, hace uso de JavaScript para obtener los dispositivos multimedia de un equipo lo cuál facilita la implementación.

En la siguiente imagen se muestra como se organizar dicha app:

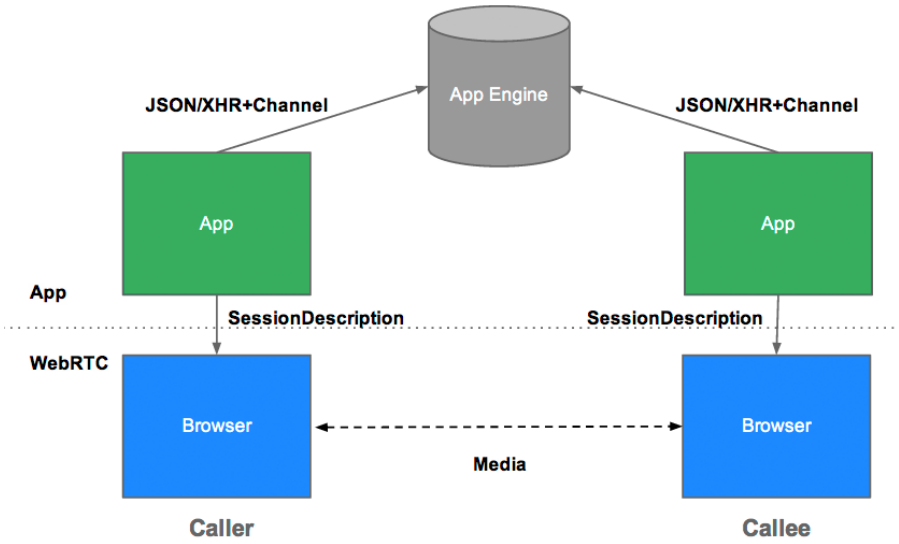


Figura 8.3.5: *Arquitectura de la aplicación de video chat de Google appRTC [31].*

Estos son los pasos para generar un identificador único que identifica a la sesión, aunque está se corte pudiendo recuperar el vídeo incluso después de un micro-corte:

1. Cliente A genera un ID único.
2. Cliente A hace una petición de token desde la App Engine, pasando su ID.
3. La App Engine responde al canal concediendo un token y canal desde la API.
4. App envía el token al Cliente A.
5. Cliente A abre un socket y escucha en el canal para configurarse.

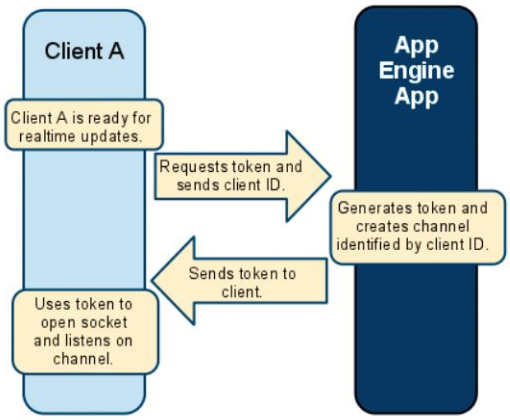


Figura 8.3.6: *La API del canal de Google: Estableciendo un canal [32].*

RTCDataChannel:

Tal y como el soporte de WebRTC permite él envió de audio/vídeo por un canal en tiempo real, también pueden enviarse otros tipos de datos.

La API *RTCDataChannel* permite el intercambio de datos punto a punto, con una pequeña latencia e incluye comprobación de errores.

En este caso el potencial uso de dicha API, incluye soporte para:

- Juegos.
- Información sobre escritorio (puntero, datos).
- Chat de texto en tiempo real.
- Transferencias de ficheros.
- Redes descentralizadas.
- La API tiene las siguientes mejoras sobre RTCPeerConnection y permite poder flexibilizar las comunicaciones peer-to-peer.
- Aprovechar la configuración de RTCPeerConnection.
- Conexión con múltiples canales simultáneamente, con priorización.
- Control de entrega y fiabilidad integradas.
- Construido para uso seguro (soportando DLS) y controles de congestión
- Permite el uso de datos con/sin video o con/sin voz.

En este caso su uso es muy similar al WebSocket, en dicho proyecto hemos hecho uso de WebSocket por ser más estable que las versiones previas de WebRTC.

8.4 Apéndice IV: Conceptos básicos para NodeJS

NodeJS es una plataforma basada en el motor de JavaScript V8 de Google utilizado en el navegador Chrome. Está pensada para facilitar el desarrollo de aplicaciones basadas en red. Utiliza un modelo I/O (entrada/salida) orientado a eventos y basado en el 'no-bloqueo', que lo hace ligero y eficiente, ideal para aplicaciones en tiempo real que hagan uso de datos intensivos y que se ejecuten a través de dispositivos distribuidos.

Vamos a ver algunas características y conceptos de esta interesante plataforma:

1. ¿Por qué JavaScript?

JavaScript es un gran lenguaje para la programación asíncrona, ya que fue diseñado para ser usado en programación orientada a eventos en lugar de otros lenguajes orientados a objetos, como, por ejemplo, Java. Es especialmente atractivo para realizar aplicaciones 'no bloqueantes' y de alta concurrencia y disponibilidad.

Al contrario que la mayoría del código JavaScript, en NodeJS este código no se ejecuta en un navegador, sino en el lado del servidor. NodeJS implementa algunas especificaciones de CommonJS. También incluye un entorno REPL para depuración interactiva.

Una aplicación para NodeJS se programa sobre un solo hilo. Si en la aplicación existe una operación bloqueante (I/O), se crea entonces otro hilo en segundo plano, pero no por cada conexión, como haría un servidor web como por ejemplo Apache o Nginx. En teoría, NodeJS puede mantener tantas conexiones como número máximo de archivos descriptores (sockets) soportados por el sistema (el número de 'MaxClients' por defecto, es 256). En un sistema UNIX este límite puede rondar las 65.000 conexiones. Sin embargo, la cifra dependerá realmente de varios factores, como la cantidad de información que una aplicación sirva a los clientes. Una aplicación 'tipo' podría mantener entre 20.000-25.000 clientes a la vez sin existir apenas retardo en las respuestas.

2. Plataforma madura y estable

NodeJS fue creado por Ryan Dahl en 2009 y su evolución está apadrinada por la empresa Joyent. La plataforma está programada en C++ y JavaScript.

Quizás en sus primeras versiones encontrábamos cierta inestabilidad. En sus comienzos era frecuente encontrar problemas con módulos de terceros, pero actualmente estos problemas están ampliamente superados.

3. Software libre

NodeJS está liberado bajo la licencia MIT y otros paquetes de licencias flexibles para componentes de software libre. Se puede descargar en forma de código fuente o instalador para Windows o Mac.

Su desarrollo por parte de la comunidad es muy activo. El código fuente está albergado en GitHub y cualquiera puede hacer un *fork* (bifurcación de código) desde allí.

Si te decides a probarlo, deberías tener en cuenta las instrucciones de instalación del entorno.

4. Ventajas

- Rapidez. Tanto en la ejecución, como en el desarrollo o la ejecución de pruebas unitarias.
- Mejora en la experiencia de usuario. Su uso proporciona una mejor experiencia en UX.
- Ahorro en infraestructura. Especialmente en el número de servidores necesarios para las aplicaciones. Por ejemplo, cuando LinkedIn migró todo su backend desde *Ruby on Rails* hacia NodeJS redujo desde 30 hasta 3 su número de servidores.
- Conjunción entre *BackEnd* y *FrontEnd*. Los equipos de BackEnd y FrontEnd móvil pueden ser combinados en un conjunto único.
- Flexibilidad.
- Fácil de usar. Sobre todo, si tienes experiencia con JavaScript.

5. Soporta multitud de protocolos y módulos

Es una plataforma bastante flexible en cuanto a los protocolos que puede manejar. Soporta los protocolos TCP, DNS y HTTP.

Hablando de módulos, Node.js pone a nuestra disposición gran cantidad de módulos propios y de terceros. Npm es la herramienta fundamental para el manejo de dependencias o módulos. Su uso no es demasiado complicado y existen tutoriales que te ayudarán en su manejo.

Una gran virtud de NodeJS, elegir bibliotecas de ‘manera arbitraria’ y combinarlas en el mismo programa sin tener que preocuparte demasiado por posibles incompatibilidades entre ellas.

6. Usos recomendados

Está especialmente indicado para desarrollar aplicaciones de los siguientes tipos:

- Aplicaciones web.
- Aplicaciones para línea de comandos y scripts para administración de sistemas.
- Aplicaciones de red.
- Juegos online.
- Gestores de correo online: de esta manera teniendo el navegador abierto podríamos ver notificaciones en tiempo real de nuevos correos recibidos.
- Herramientas de colaboración.
- Chats.
- Redes sociales.
- Herramientas de traducción en tiempo real.
- Confección de APIs: APIs en lado del servidor para proporcionar rendimiento y escalabilidad. Debido a que permite ejecutar gran cantidad de peticiones.
- Aplicaciones web que necesiten una conexión persistente con el navegador del cliente. Mediante una serie de técnicas llamadas *Comet*, puedes hacer una aplicación que envíe datos al usuario en tiempo real; es decir, que el navegador mantenga la conexión siempre abierta y reciba continuamente nuevos datos.

7. ¿Posibles inconvenientes en su uso?

Por la propia naturaleza de NodeJS, se deben realizar las peticiones de manera asíncrona, podría haber ciertas ocasiones en las que no interesara hacerlo de esta manera. Si se quiere trabajar de manera síncrona, entonces estaríamos 'obligados' a instalar algunos módulos que de momento están poco maduros.

Debido a su arquitectura en la que se usa un sólo un hilo, sólo puede usar una CPU. Un método para usar múltiples núcleos sería iniciar múltiples instancias de Node.js en el servidor y poner un balanceador de carga delante de ellos.

8. ExpressJS como framework

No es solamente un *framework* de gran calidad pare el uso de NodeJS. Estamos ante el que es para muchos uno de los 'marcos de trabajo' más flexibles y fáciles de utilizar dentro de los aparecidos en los últimos tiempos.

Las aplicaciones desarrolladas con ExpressJS son de una calidad indudable. Y para muestra un botón: Klout, Storify, Prismatic o la nueva versión de MySpace.

9. Comunidad que apoya y mejora NodeJS

Node.js tiene una comunidad muy numerosa y activa. Desde su web tienes acceso a la documentación y API, tutoriales, foros o un útil blog.

10. Recursos

Una buena manera de familiarizarte con NodeJS aparte de utilizar los recursos que nos proporciona Internet es el libro 'Node.js in Action' de la conocida editorial técnica Manning.

Uno de los puntos fuertes de NodeJS es la disponibilidad de una gran cantidad de librerías y otros recursos interesantes. Estos son sólo algunos ejemplos:

- Socket.IO
- Expressjs
- node.dbslayer.js
- Handlebars.js

11. Cada vez más demandado

No siempre es así, pero la importancia de un entorno, plataforma, *framework* o lenguaje de programación, se suele medir en la demanda profesional que suscita. Con respecto a NodeJS, cada vez hay una mayor demanda de profesionales con experiencia en esta plataforma. Solamente hay que darse una vuelta por los sitios más conocidos de reclutamiento en IT para darse cuenta. Desde la propia web de NodeJS manejan una bolsa de trabajo específica.

12. Pero, ¿quién lo utiliza?

Como hemos vista en el punto anterior, su demanda está aumentando en las empresas. Desde startups a empresas consolidadas como Yahoo!, Microsoft, LinkedIn o eBay.

8.5 Apéndice V: Detección de errores con Splunk Mint

MINT es una herramienta para el desarrollo de aplicaciones móviles muy útil. Antiguamente se le conocía como *BugSense*. Su uso se basa en captar información sobre los errores que se producen en ejecución cuando los usuarios usan nuestra aplicación y enviarlos a la nube de Splunk donde nosotros como desarrolladores podremos revisarlos, para solventar los posibles errores en diferentes configuraciones y terminales.

La web del proyecto es: http://www.splunk.com/en_us/products/splunk-mint.html

Aunque tienen una versión de pago para empresas, en nuestro caso hemos hecho uso de la versión Gratuita que cumple con todas las garantías para el uso estipulado.

Los datos que hace disponible la herramienta, a la que se accede mediante una página web, son los siguientes:

- Fallos inesperados de la aplicación.
- Fallos manejados de la aplicación.
- Versión del sistema operativo en la que ha ocurrido el fallo.
- Versión de la aplicación en la que ha ocurrido el fallo.
- Tipo de fallo y su traza.
- Grafica de los fallos ocurridos en los últimos siete días.

Instalacion en nuestro proyecto:

Su instalación a cualquier proyecto es realmente sencilla, necesitando solamente una librería .jar para añadir a nuestro proyecto. Declarar el uso de dicha librería en el build.graddle y después en nuestro proyecto lanzar y parar dicha librería.

1º Copiamos la librería a nuestro proyecto en la ruta:
..\UMA-AndroidWebRTC\app\libs\splunk-mint-4.3.0.jar

2º Declaramos el uso de la librería en el fichero:
..\UMA-AndroidWebRTC\app\build.gradle

```
dependencies {
    compile files('libs/libjingle_peerconnection-7113.jar')
    compile files("libs/lib_android_websockets.jar")
    compile files('libs/splunk-mint-4.3.0.jar')
}
```

Ya estamos preparados para el uso de dicho plugin, sólo nos falta lanzarlo en nuestro proyecto; en este caso lo lanzaremos en la actividad MainActivity y lo finalizaremos a la salida de la aplicación.

En el método *OnCreate* de la clase *MainActivity* después de cargar el *layout* añadimos dichas líneas (“9e84fc32” es nuestra *API-KEY* obtenida al registrarnos en la web!)

```
Mint.disableNetworkMonitoring();// No funcione sino hay internet
Mint.initAndStartSession(this, "9e84fc32"); // Lanzamos el servicio de
Debugging Mint
```

Para terminar de monitorizar, en nuestro caso a la salida de la aplicación controlada por un AlertDialog para preguntarnos si deseamos cerrar la app.

```
preferencias.GuardarPreferencias(); // Guardamos que es gratis :P
      final AlertDialog show = new
Builder(this).setIcon(android.R.drawable.ic_dialog_alert).setTitle("Salida")
            .setMessage("¿Quieres cerrar la ventana?")
            .setPositiveButton("Si", new DialogInterface.OnClickListener() {
               @Override
               public void onClick(DialogInterface dialog, int which) {
                  Mint.closeSession(MainActivity.this);
                  // wakeLock.release();
                  System.exit(-1);
                  //finish();}
            }).setNegativeButton("No", null).show();
   } // fin else
} // fin pulsar atrás

Mint.closeSession(MainActivity.this) ; // usado al salir de la aplicación
```

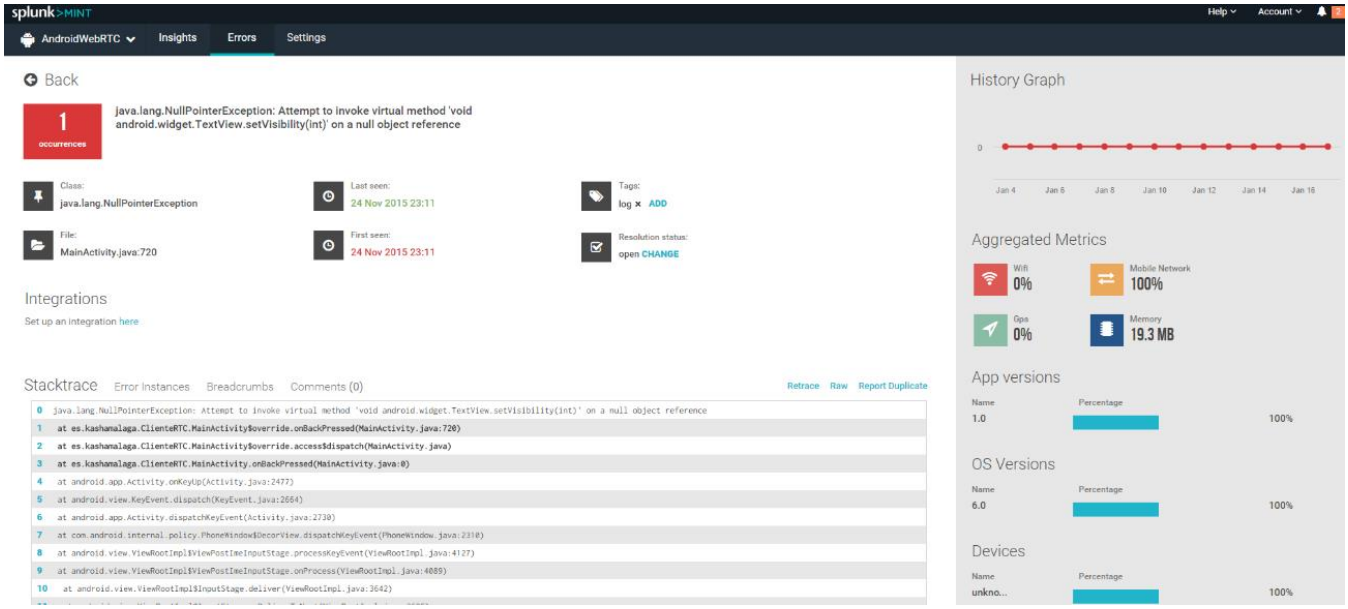


Figura 8.5.1: *Ventana de la Web MINT para la depuración de la aplicación móvil [33].*

En la imagen superior puede verse la traza de un error detectado en un dispositivo que ejecutó nuestra aplicación.

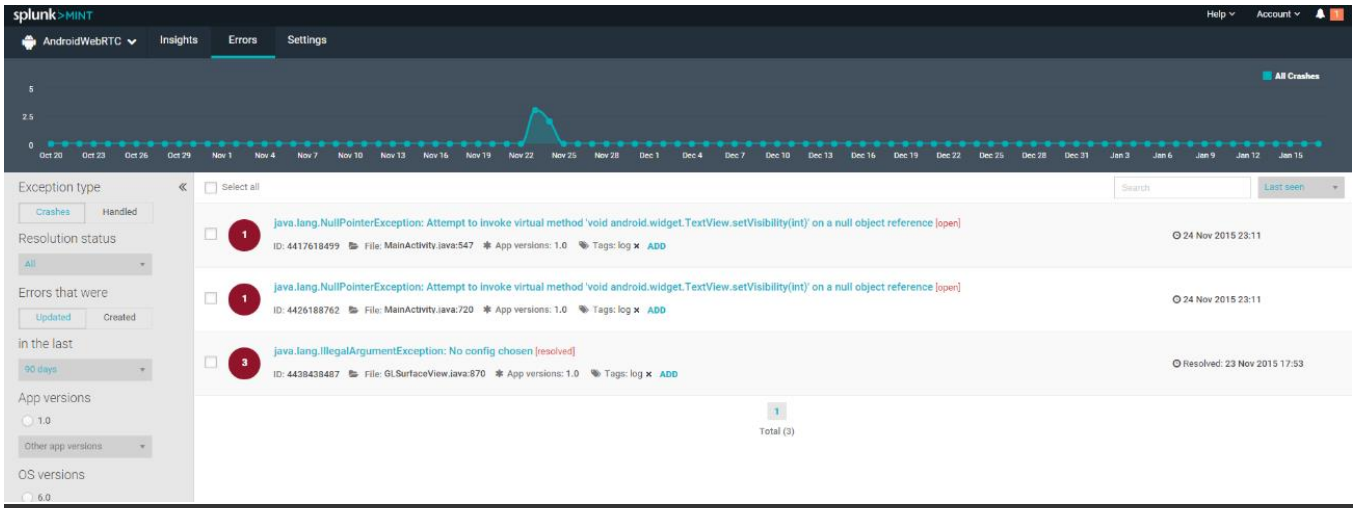


Figura 8.5.2: *Diversos errores capturados a lo largo de 90 días [34].*