



UNIVERSIDAD DE CASTILLA-LA MANCHA
ESCUELA SUPERIOR DE INFORMÁTICA

MÁSTER UNIVERSITARIO EN INGENIERÍA
INFORMÁTICA

TRABAJO FIN DE MÁSTER

Diseño e implementación de un sistema basado en realidad
aumentada para la teleasistencia
en situaciones de emergencia

Francisco Manuel García Sánchez-Belmonte

diciembre, 2022



**UNIVERSIDAD DE CASTILLA-LA MANCHA
ESCUELA SUPERIOR DE INFORMÁTICA**

**DEPARTAMENTO DE TECNOLOGÍAS Y SISTEMAS DE
INFORMACIÓN
TRABAJO FIN DE MÁSTER**

**Diseño e implementación de un sistema basado en
realidad aumentada para la teleasistencia
en situaciones de emergencia**

Autor: Francisco Manuel García Sánchez-Belmonte

Tutor: Carlos González Morcillo

Tutor: David Vallejo Fernández

diciembre, 2022

Diseño e implementación de un sistema basado en realidad aumentada para la teleasistencia en situaciones de emergencia

© Francisco Manuel García Sánchez-Belmonte, 2022

Este documento se distribuye con licencia CC BY-NC-SA 4.0. El texto completo de la licencia puede obtenerse en <https://creativecommons.org/licenses/by-nc-sa/4.0/>.

La copia y distribución de esta obra está permitida en todo el mundo, sin regalías y por cualquier medio, siempre que esta nota sea preservada. Se concede permiso para copiar y distribuir traducciones de este libro desde el español original a otro idioma, siempre que la traducción sea aprobada por el autor del libro y tanto el aviso de copyright como esta nota de permiso, sean preservados en todas las copias.

Este texto ha sido preparado con la plantilla \LaTeX para Trabajo Fin de Estudios en Ingeniería Informática para la UCLM publicada por [Jesús Salido](#) en el repositorio público Zenodo, DOI: [10.5281/zenodo.4561708](https://doi.org/10.5281/zenodo.4561708), como parte del curso « *\LaTeX esencial para preparación de TFG, Tesis y otros documentos académicos*» impartido en la Escuela Superior de Informática de la Universidad de Castilla-La Mancha. Si te ha resultado de utilidad te agradeceré que la cites [?] e incluyas en tus referencias como se indica en Zenodo con el DOI suministrado para todas las versiones.



TRIBUNAL:

Presidente: _____

Vocal: _____

Secretario(a): _____

FECHA DE DEFENSA: _____

CALIFICACIÓN: _____

PRESIDENTE

VOCAL

SECRETARIO(A)

Fdo.:

Fdo.:

Fdo.:

*"Por fin...
...otra vez."*

Diseño e implementación de un sistema basado en realidad aumentada para la teleasistencia en situaciones de emergencia

Francisco Manuel García Sánchez-Belmonte
Ciudad Real, diciembre 2022

Resumen

La realidad mixta resulta de la combinación de realidad virtual y aumentada para unir de forma interactiva y creíble un entorno físico con otro generado por computador. Este trabajo de fin de máster propone aprovechar dicha fusión de mundos y usarla como pilar central para el diseño y desarrollo de un sistema con el fin facilitar y apoyar las tareas de asistencia llevadas a cabo por unidades de emergencia.

Imagínese un escenario donde el personal de un equipo de emergencias porta unas gafas de realidad aumentada conectadas a la red 5G las cuales transmiten de forma bidireccional flujos de audio, vídeo y datos a un médico especialista que se encuentra a una distancia indeterminada. Se pueden apreciar las grandes ventajas que traería la integración de realidad mixta en este tipo de circunstancias: el especialista tendría la capacidad de acceder en directo al punto de vista del personal de la unidad móvil y ofrecer indicaciones orales y visuales, incluyendo gestos y el posicionamiento de marcadores gráficos en el espacio tridimensional, a estos últimos con el fin de que puedan poner en marcha los mejores tratamientos posibles para el paciente sin tener que esperar a que este llegue al centro médico ahorrando mucho tiempo en escenarios donde cada segundo puede marcar una gran diferencia.

Finalmente, la arquitectura software diseñada se generalizará para que sirva como base para el despliegue del sistema en entornos en los que no sea posible hacer llegar una unidad de emergencias, pero sí un dron o cualquier otro tipo de transporte con el que se pueda hacer entrega de los dispositivos apropiados a los afectados.

Design and implementation of an augmented reality-based system for telecare in emergency situations

Francisco Manuel García Sánchez-Belmonte
Ciudad Real, December 2022

Abstract

Mixed reality arises from the combination of virtual and augmented reality to interactively and believably merge a physical environment with a computer-generated one. This master's thesis proposes to take advantage of this fusion of worlds and use it as a central pillar for the design and development of a system in order to facilitate and support the assistance operations carried out by emergency units.

Consider a scenario where emergency personnel wear augmented reality glasses connected to the 5G network which can transmit audio, video and data streams bidirectionally to a medical specialist at an arbitrary distance. It's possible to see the great advantages the integration of mixed reality would bring in this type of circumstances: the specialist would have the ability to access the point of view of the mobile unit staff at any time and offer oral and visual indications, including gestures and positioning of graphic markers in three-dimensional space so they can provide the best possible care to the patient without having to wait for the him or her to arrive at the medical facility, saving a lot of time in scenarios where every second can make a huge difference.

Finally, the proposed software architecture will be generalised to serve as the foundation for the deployment of the system in environments where an emergency unit can't reach but is feasible to send a drone or any other type of vehicle to deliver the appropriate devices to the victims.

Agradecimientos

En primer lugar agradecer a mi familia, y en concreto a mis padres y abuelos, su apoyo constante durante todos estos años. Nunca me ha gustado mucho estudiar y si no hubiera sido por su paciencia y dedicación quizás no habría acabado este máster (ni el instituto, seguramente).

En segundo lugar a Ana y Ainhoa, que han intentado evitar (con éxito) que me tire por la ventana durante la realización de este proyecto.

En tercer lugar a todos y cada uno de los amigos que he hecho durante mi paso por la ESI. No voy a poner vuestros nombres porque sois un montón y si se me olvida alguien seguro que me lo está recordando hasta el día en que me muera.

Y por ultimo, pero por supuesto no menos importante, a mis tutores Carlos y David por dirigir este proyecto y al resto de compañeros de Furious Koalas. Hacéis que trabajar no me cueste trabajo.

Francisco Manuel García Sánchez-Belmonte
Ciudad Real, 2022

Índice general

Índice de figuras

Índice de tablas

Índice de listados

Introducción

En los últimos años están apareciendo cada vez más publicaciones y trabajos relativos a la realidad mixta (RM), la cual resulta de la combinación de la realidad aumentada (RA) con la realidad virtual (RV).[?] Sí bien en la realidad aumentada tradicional la información sintética será simplemente superpuesta sobre el mundo real, en la realidad mixta se integrará completamente el entorno virtual y el físico con el fin de así permitir interacciones naturales e intuitivas entre los usuarios y un mundo virtual generado por computador.

Con el paso del tiempo la RM está teniendo un papel mayor en nuestra sociedad, hasta tal punto que se está aplicando en campos tan diversos como los videojuegos, industria, marketing, educación e incluso salud. Esta tecnología permite cambiar el paradigma de atención sanitaria actual, acelerando los diagnósticos y reduciendo el tiempo de atención, aumentando la personalización y mejorando los resultados.

Como tecnología emergente, la realidad mixta tiene un gran potencial en el área de la telemedicina, la cual se define como el uso de las tecnologías de la información y las telecomunicaciones para dar servicios médicos allí donde sea necesario.[?] La utilidad de la telemedicina resalta en aquellos entornos donde no pueden acceder o no estén presentes los médicos con adecuada formación, por ejemplo un vuelo comercial, un campo de batalla o en el entorno doméstico.

El caso de uso que se trata en este proyecto propone facilitar y apoyar las tareas de asistencia que llevan a cabo unidades de emergencia desplegadas en diversos lugares mediante el desarrollo de una serie de herramientas hardware y software. El contexto en el que surge esta idea se enmarca en una propuesta de la *spin-off* de la UCLM Furious Koalas S.L. para la segunda convocatoria de subvenciones a proyectos piloto de tecnologías 5G en el que participan los siguientes actores:

- **Furious Koalas:** aporta su experiencia y conocimiento sobre servicios integrales de alta calidad en las áreas de representación gráfica interactiva y el vídeo interactivo además de ser el coordinador del caso de uso.
- **Telefónica:** colabora en este caso de uso poniendo a disposición la red 5G en el área de Talavera de la Reina, y la banda de frecuencias que se va a utilizar en él, es decir, la de 3,5 GHz.
- **Hospital Nuestra Señora del Prado (Talavera de la Reina):** pone su experiencia sanitaria para detallar los requisitos operativos y funcionales del caso de uso, y personal médico para realizar la prueba del caso de uso. Es usuario final del caso de uso.
- **Gerencia de Urgencias, Emergencias y Transporte Sanitario del SESCAM (GUETS):** se ocupa de aportar su experiencia sanitaria para detallar los requisitos operativos y funcionales del caso de uso, así como un vehículo de atención a emergencias con su equipo humano para realizar las pruebas pertinentes. Es el usuario final del caso de uso.

La RM posibilitará la integración, en tiempo real, de objetos virtuales 3D sobre un entorno real. Así, la realidad mixta en este caso de uso permitirá que las personas de la unidad médica de emergencia visualicen protocolos de actuación virtuales en base a las indicaciones y actuaciones realizadas por los médicos especialistas de forma remota. Las acciones o recomendaciones virtuales

asociadas a estos protocolos se integrarán en el mundo real gracias a la utilización de unas gafas de RA. De este modo, el usuario del sistema de realidad aumentada podrá ver, perfectamente alineada sobre su percepción del mundo real, imágenes virtuales generadas por ordenador que le ayudarán a realizar las tareas de asistencia médica. Estas indicaciones podrán ser objetos 3D de diferentes formas e incluso dibujados a mano alzada.



Figura 1.1: Personal de emergencia desplegado a distancia (izquierda) recibiendo instrucciones de un médico especialista (derecha)

En la parte izquierda de la imagen ?? se muestra gráficamente cómo el personal de un equipo móvil de emergencias cuenta con unas gafas de realidad aumentada conectadas a la red 5G y que transmiten en tiempo real un flujo de audio vídeo y datos. En la parte derecha de la imagen se puede ver al médico especialista que ubicado a distancia recibe estos medios y ofrece indicaciones de forma remota al personal de emergencias.

A modo de ejemplo, la parte derecha de la figura ?? refleja gráficamente, y a alto nivel, las interacciones que se producen entre el personal del equipo móvil y el médico especialista: 1) observación del paciente, 2) comunicación del equipo móvil con el médico especialista, 3) *feedback* ofrecido por el médico especialista en base a la percepción del mundo real enviada por el sistema, 4) aplicación del protocolo médico correspondiente. Más allá de transmitir el vídeo y el audio sobre la infraestructura 5G, en la imagen se aprecia información aumentada sobre la visualización del mundo real que hace la persona que porta el dispositivo de realidad aumentada. Por otro lado, la parte izquierda de la figura muestra cómo el sistema añadiría, de nuevo, información virtual sobre el flujo de vídeo que el médico especialista recibe del personal de emergencias móviles. Este esquema facilitaría la comunicación y la interacción entre los dos extremos.



Figura 1.2: Esquema general de la interacción entre los participantes

Se pueden ver rápidamente las grandes ventajas que trae la integración de RM en este tipo de circunstancias: el especialista podrá dar indicaciones orales y visuales precisas para tratar al paciente sin tener que esperar a que este llegue al centro médico, ahorrando mucho tiempo en situaciones donde cada segundo puede marcar una gran diferencia.

Finalmente, se podría extender el caso de uso para reutilizar estas herramientas en entornos donde no sea posible hacer llegar un equipo de emergencias pero sí un dron o cualquier otro tipo de transporte con el que hacer entrega de los dispositivos apropiados a los afectados.



Figura 1.3: Transporte de un dispositivo de realidad aumentada por parte de un dron con el fin de prestar asistencia médica remota

El carácter disruptivo del caso de uso reside en incorporar un esquema de interacción natural basado en realidad mixta en situaciones de emergencia médica donde el personal móvil requiere la asistencia de un médico especialista en tiempo real. Esta asistencia se sustenta en una comunicación audiovisual de baja latencia que elimina la barrera de la distancia física, consiguiendo un efecto similar al que se podría conseguir en el caso de disponer del médico especialista de forma presencial en la emergencia médica.

CAPÍTULO 2

Objetivos

2.1. OBJETIVO GENERAL

El objetivo general de este proyecto es el diseño y desarrollo de un sistema distribuido compuesto por múltiples aplicaciones que dotará a un equipo de emergencias desplegado en el lugar de un accidente con la capacidad de comunicarse en tiempo real mediante voz, vídeo y datos con un médico especialista ubicado en un hospital. Los requisitos más importantes que cumplir serán los siguientes: la aplicación destinada al equipo de emergencias tiene que implementar un paradigma de interacción basado en realidad mixta y se deberán poder añadir dispositivos médicos al sistema cuyos datos serán accesibles por parte de todos los participantes en la conferencia.

2.2. OBJETIVOS ESPECÍFICOS

Estos objetivos generales pueden ser desglosados en diferentes objetivos más concretos que se expondrán a continuación.

- **Estudio del estado del arte en cuanto a las interfaces gráficas de usuario para sistemas de realidad aumentada:** se deberá llevar a cabo una revisión de diferentes aplicaciones de temáticas relacionadas con la de este proyecto.
- **Estudio y análisis de diferentes tecnologías de comunicación en tiempo real con el fin de realizar la elección de la más adecuada para el proyecto:** la tecnología de comunicación utilizada deberá permitir el intercambio de audio, vídeo y datos de manera bidireccional, tener baja latencia y estar soportada tanto por un computador de escritorio como por el *headset* HoloLens 2.
- **Diseño, implementación, despliegue y pruebas de una arquitectura consistente en cuatro aplicaciones de dos temáticas diferenciadas:**
 - **Videoconferencia:** engloba las aplicaciones destinadas al médico especialista y al equipo de emergencias.
 - **Integración con dispositivos médicos:** solución compuesta por una parte a ejecutarse cerca del equipo de emergencias y otra en la nube. Permitirá al médico especialista y al equipo de emergencias disponer de datos vitales del paciente en tiempo real.

Metodología

De entre todas las alternativas posibles, la metodología empleada para llevar a cabo la implementación de este sistema ha sido basada en el Desarrollo Rápido de Aplicaciones (*RAD, Rapid Application Development*)[?], debido a que se ha considerado interesante tener disponibles prototipos funcionales de los programas lo antes posible en el ciclo de desarrollo.

El proceso ha consistido primero en la elicitación de los requisitos que debe cumplir el sistema y que se ven expuestos en el capítulo de objetivos de este documento. Aparte de los objetivos hay requisitos más concretos que debe satisfacer el programa entre los que podemos citar como el tener una optimización adecuada, posibilitar una comunicación fluida y permitir el empleo de múltiples marcadores en el espacio tridimensional.

Una vez con los requisitos claros se ha comenzado a prototipar. El desarrollo del proyecto ha sido dividido en cuatro hitos o prototipos diferenciados, siendo los tres primeros los que se indican a continuación:

1. **Establecimiento de la videoconferencia:** se inicia el desarrollo con la implementación de un sistema de comunicación bidireccional de vídeo y audio en tiempo real y de baja latencia, compuesto por una aplicación para HoloLens 2 para el equipo de emergencias y otra para computador de escritorio para el médico especialista.
2. **Posicionamiento de marcadores en el espacio:** el objetivo principal de este hito es dar la posibilidad al médico especialista de colocar un marcador en el espacio tridimensional que rodea al usuario de HoloLens 2 haciendo clic en el vídeo obtenido desde la cámara del *headset*.
3. **Múltiples marcadores y herramientas:** se añaden diferentes tipos de marcadores y herramientas de interacción con los mismos, así como la posibilidad de escalarlos, moverlos y borrarlos. Especial mención de este hito a la implementación de un pincel que permite al médico especialista pintar sobre el espacio tridimensional.

Finalmente, el último hito incluye la integración de dispositivos médicos y la corrección de *bugs* de última hora que se hayan ido detectando. Se han introducido nuevas formas de interacción en la aplicación destinada al equipo de emergencias como la manipulación de los marcadores mediante gestos y los comandos de voz. Otras características muy importantes de esta fase son la generación de la mayoría de documentación relativa al proyecto usando la herramienta Doxygen y la escritura de esta memoria.

3.1. HARDWARE Y SOFTWARE UTILIZADO EN EL PROYECTO

En esta sección se introducirá el hardware y el software que ha sido necesario para la realización de este proyecto.

3.1.1. Medios hardware

Los medios hardware involucrados en el desarrollo de este proyecto han sido proporcionados por Furious Koalas S.L. y serán citados a continuación:

- MSI MS-16RU: computador portátil donde se llevará a cabo el desarrollo del proyecto así como las demostraciones oportunas.
- Microsoft HoloLens 2¹: *headset* de realidad mixta utilizado en el proyecto.
- TP-Link Archer AX10: router WiFi necesario para conectar el dispositivo HoloLens 2 a la red.

3.1.2. Medios software

El entorno software ha estado compuesto por los siguientes programas:

- Windows 10 Pro: sistema operativo elegido por ser imprescindible para trabajar con el headset HoloLens 2.
- Blender²: *suite* de código abierto para el modelado en 3D en el que se han creado los diferentes marcadores utilizados en el proyecto.
- Unity³: motor de videojuegos seleccionado como base para el desarrollo de este proyecto.
- Microsoft Visual Studio Community 2019⁴: IDE utilizado para escribir las partes de código C# de la aplicación.
- Doxygen⁵: herramienta de generación automática de documentación a partir de ficheros de código fuente con comentarios.

3.1.3. Lenguajes de programación y bibliotecas

- C#: lenguaje de programación orientado a objetos desarrollado y estandarizado por Microsoft con sintaxis derivada de C/C++ y un modelo de objetos basado en el de Java. Se usará este lenguaje puesto que es el que mejor soporte ofrece por parte de Unity.
- Python: lenguaje de programación utilizado en las aplicaciones de integración con dispositivos médicos.
- Microsoft Mixed Reality Toolkit⁶: proyecto de Microsoft para ofrecer una serie de componentes y características con el fin de acelerar el desarrollo de aplicaciones de realidad mixta.

¹<https://learn.microsoft.com/es-es/hololens/hololens2-hardware>

²<https://www.blender.org/>

³<https://unity.com/es>

⁴<https://visualstudio.microsoft.com/es/vs/community/>

⁵<https://www.doxygen.nl/>

⁶<https://github.com/microsoft/MixedRealityToolkit-Unity>

Estado del Arte

4.1. REALIDAD AUMENTADA Y TELEMEDICINA

Existe una cantidad considerable de trabajos publicados hasta la fecha en los que se hace uso de la realidad mixta, aumentada o virtual en el campo de la telemedicina. A lo largo de este punto se hará una revisión de algunos cuya temática está relacionada con el caso de uso desarrollado en este trabajo de fin de máster.

4.1.1. A novel augmented reality navigation system for endoscopic sinus and skull base surgery: a feasibility study

Este trabajo propone un sistema de navegación para cirugías endoscópicas de senos paranasales basado en realidad aumentada en el que imágenes tridimensionales obtenidas a partir de tomografías o resonancias magnéticas se fusionarán con la imagen real proveniente de la cámara de un endoscopio.[?] A continuación se muestra la interfaz de usuario de este sistema:.

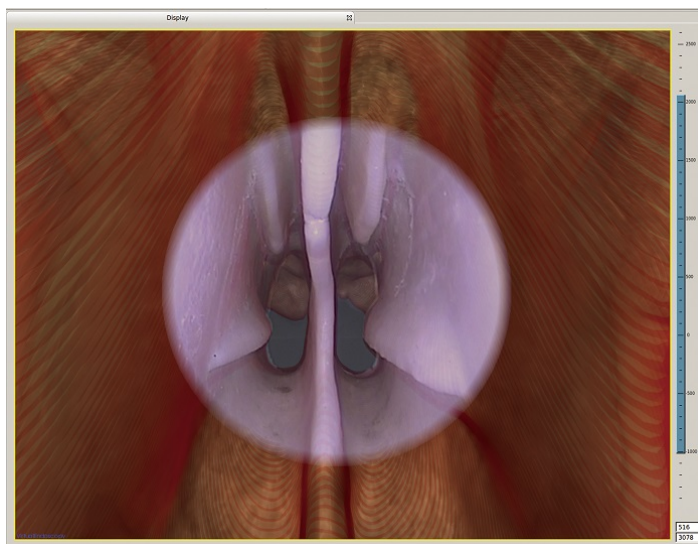


Figura 4.1: Superposición de imágenes endoscópicas reales con un modelo tridimensional[?]

Se puede ver la imagen real (en el centro) superpuesta de forma precisa sobre el modelo 3D generado mediante diferentes pruebas. Este modo de visualización aporta una nueva perspectiva a los rinólogos, pudiendo así no solo observar la imagen real del paciente, sino ponerla en contexto con todas las estructuras anatómicas que las rodean, permitiendo cirugías más seguras y reduciendo el esfuerzo mental por parte del especialista médico, además de disminuir de manera significativa su tiempo de respuesta.

4.1.2. Telemedicine supported by augmented reality: an interactive guide for untrained people in performing an ECG test

Un problema que trae consigo el uso de la telemedicina (sobretudo en situaciones de emergencia) es la de proporcionar unos cuidados adecuados cuando el usuario final no es profesional médico. En este contexto surge el siguiente trabajo en el cual se mide el desempeño de unos usuarios con baja o nula formación médica a la hora de realizar un electrocardiograma (ECG) utilizando unas gafas de realidad aumentada como medio para guiarlos.[?]

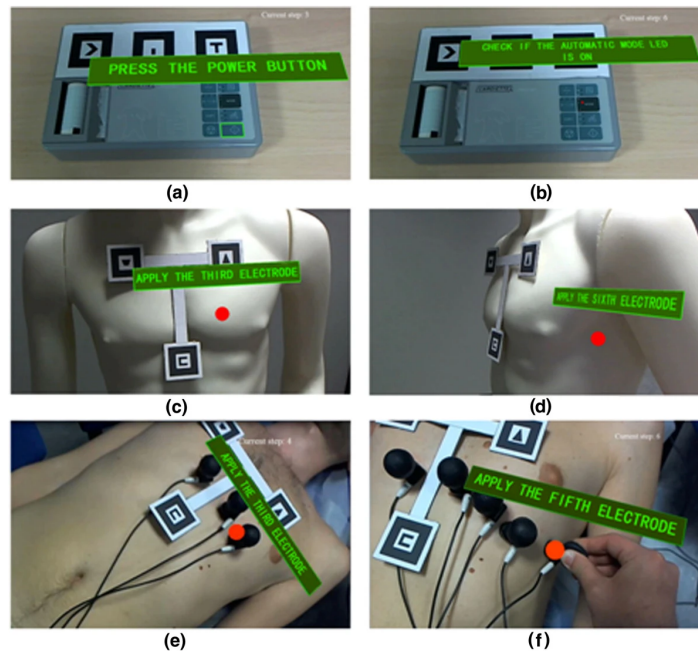


Figura 4.2: Diferentes escenas aumentadas usando marcadores[?]

Se puede ver en la imagen ?? que la aplicación desarrollada orienta al usuario en las acciones que debe realizar a través de diferentes marcadores virtuales ubicados con la ayuda de algunos marcadores físicos, además el sistema cuenta con comandos de voz para repetir o avanzar entre las instrucciones. El proyecto fue un éxito pues todos los participantes pudieron completar las pruebas en un tiempo aceptable y sin encontrar dificultades particulares en el uso del dispositivo de realidad aumentada.

4.1.3. Augmented reality as a telemedicine platform for remote procedural training

En el siguiente estudio se presenta un sistema basado en HoloLens (primera versión del *headset*) que permite el establecimiento de una videoconferencia y la intervención remota del médico experto a través de un modelo 3D de una mano que imita los gestos que hace este último.[?]



Figura 4.3: Videoconferencia con el punto de vista de cada participante[?]

En la parte izquierda de la imagen ?? se ve al especialista realizando un gesto que después será superpuesto en el espacio tridimensional de la persona que lleva a cabo el procedimiento médico gracias a la realidad aumentada. En general se obtuvieron resultados positivos, más este estudio cuenta con algunas limitaciones que reseñan los autores, como el uso de una red local por problemas con el ancho de banda, ciertas incomodidades a la hora de usar las gafas o la escasa documentación para el desarrollo de aplicaciones de HoloLens 1.

4.1.4. Augmented reality assisted surgery: a urologic training tool

Este estudio propone una aplicación basada en realidad aumentada y videoconferencia en tiempo real con el fin de asistir en operaciones para la colocación de prótesis inflables de pene usando un *headset* óptico, en concreto Google Glass.[?]

Esta aplicación dispone de varias funciones importantes: la primera de ellas es mostrar un vídeo interactivo paso a paso del proceso proyectado sobre la imagen del paciente en tiempo real, mientras que la otra es establecer una videoconferencia con personal especialista ubicado a distancia.

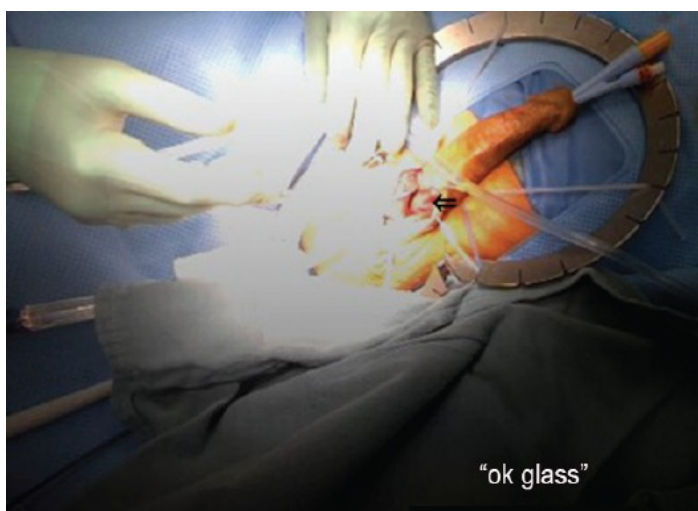


Figura 4.4: Colocación de puntos de interés sobre imagen real[?]

En la imagen ?? se muestra como el personal especialista puede interactuar con la imagen real obtenida desde las gafas en base al posicionamiento de marcadores sobre la misma, lo que permite indicar al cirujano puntos de interés destacados con el fin de orientar mejor los procedimientos.

4.1.5. Avoiding focus shifts in surgical telementoring using an augmented reality transparent display

El trabajo de Andersen *et al.* propone un sistema de realidad aumentada para evitar la pérdida de concentración que se produce cuando un estudiante de cirugía debe dejar de prestar atención al proceso que está realizando para mirar un monitor u otro medio con el fin de recibir indicaciones por parte de su mentor.[?]



Figure 2. Our surgical telementoring prototype system, showing the trainee module (*left*), the mentor module (*center*), and the trainee's first-person view of the operating field through the trainee module (*right*).

Figura 4.5: Estudiante realizando un entrenamiento con anotaciones basadas en AR[?]

Se puede observar en la imagen ?? como el estudiante tiene en todo momento dentro de su campo visual una *tablet* que actúa a modo de “pantalla transparente” al mostrar la imagen real de lo que ocurre debajo de la misma. Esta imagen real será enviada al profesor para que sobre ella pueda colocar anotaciones o indicar procedimientos que serán mostrados al estudiante, evitando así que este último se vea obligado a mirar a otra parte si necesita ayuda.

4.1.6. Virtual interactive presence in global surgical education: international collaboration through augmented reality

A continuación se expone otro ejemplo de proyecto en el que se aplica la realidad aumentada y la videoconferencia a la telemedicina: una herramienta basada en una aplicación para iPad que permite a cirujanos de centros hospitalarios lejanos colaborar entre sí en tiempo real durante una intervención.[?]



Figura 4.6: Vista desde quirófano (izquierda), punto de vista del médico especialista (derecha)[?]

En este proyecto las imágenes reales obtenidas por parte del equipo de quirófano son enviadas a través de la red a un médico experto que se encuentra a distancia y compuestas con el flujo de imagen obtenido desde la cámara del *tablet* de este último, permitiendo una comunicación visual compleja. Esta arquitectura se ha utilizado con éxito en 15 operaciones con personal simultáneamente en Vietnam y EEUU sin ninguna complicación significativa.

4.1.7. Disaster medicine through Google Glass

La medicina de desastres (*disaster medicine* en inglés) es otra área relacionada con la telemedicina en la que se han publicado trabajos basados en realidad mixta. *Disaster medicine through Google Glass* hace uso del dispositivo de Google para guiar de forma visual en el triaje de las víctimas de un desastre y recoger toda la información posible del entorno, lo que será usado más tarde para enfocar los recursos a donde sean de mayor utilidad.[?]

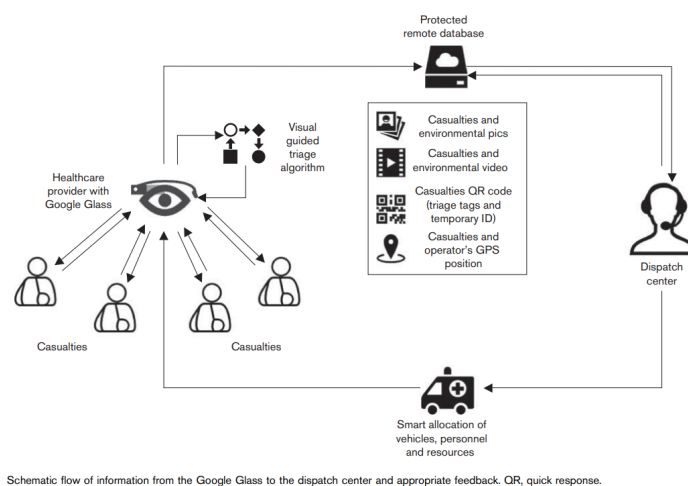


Figura 4.7: Flujo de la información desde su generación hasta su uso por parte del centro de emergencias[?]

Se puede ver en la figura ?? un esquema del funcionamiento del sistema, en el cual el operador que lleva las gafas de realidad aumentada deberá acercarse a un paciente y responder a una serie de preguntas que el dispositivo le hará sobre el estado del mismo. Esta información, junto con el audio, vídeo, imágenes estáticas y coordenadas GPS serán registradas en una base de datos remota para ser usadas por parte del centro de mando para hacer una asignación inteligente de los recursos médicos disponibles.

4.1.8. Technical support by smart glasses during a mass casualty incident: a randomized controlled simulation trial on technically assisted triage and telemedical app use in disaster medicine

Sí bien existen varios algoritmos para llevar a cabo el triaje de un paciente en situaciones de emergencia, no siempre son usados de la manera correcta. En este contexto surge este trabajo que utiliza Google Glass y realidad aumentada para realizar diferentes preguntas con el fin de categorizar a las víctimas de un desastre y compara la eficacia de este método con el tradicional.[?] En la figura ?? aparece una captura de pantalla de la interfaz de usuario del sistema preguntando si el paciente está inconsciente o no:



Figura 4.8: Captura de pantalla de la interfaz de usuario de este proyecto[?]

Los resultados del estudio fueron sorprendentes: mientras que los integrantes del grupo de control realizando los triajes sin asistencia obtuvieron una precisión del 58 %, este número se elevó hasta el 92 % en el caso de los que usaron realidad aumentada para su labor.

4.2. COMUNICACIÓN EN TIEMPO REAL

A la hora de crear una infraestructura de comunicaciones hay que conocer de qué alternativas se dispone, siendo estas la comunicación síncrona o en tiempo real y la comunicación asíncrona o en diferido. La elección final dependerá fuertemente de la clase de proyecto que se esté llevando a cabo y los objetivos deseados del mismo.

Se conoce como comunicaciones en tiempo real a las telecomunicaciones en las que dos o más usuarios pueden compartir entre sí cualquier tipo de información de forma aparentemente instantánea. Dicho de otro modo, la tecnología permite que los datos sean transmitidos de una forma en la que la latencia y otros retrasos son lo suficientemente reducidos para lograr una experiencia similar a una conversación cara a cara, contando con la ventaja de facilitar la colaboración en tiempo real y la aportación instantánea de retroalimentación por parte de los implicados.[?]

Además, podemos distinguir entre comunicación en tiempo real dúplex y semidúplex, siendo la primera en la que puede existir un intercambio de datos bidireccional y simultáneo mientras que en el segundo tipo la información podrá transmitirse de forma bidireccional pero no simultáneamente.

En las comunicaciones asíncronas se prevé un retraso entre la transmisión y la recepción de información a la vez que no existe un camino directo entre origen y destino, sino que los datos que envía un emisor se almacenan durante un tiempo indefinido hasta que un emisor la recibe.

Trasladando esto a ejemplos mundanos: podemos pensar en las comunicaciones en tiempo real como una llamada de teléfono en la que las señales eléctricas que codifican el audio llegan sin un retraso apreciable (salvo problemas con la línea), mientras que las comunicaciones asíncronas son equivalentes al correo ordinario: una carta es almacenada en la oficina de correos hasta que el cartero la recoge y la entrega al destinatario.

Se concluye rápidamente que en un sistema como el presente, en el que audio y vídeo tienen que fluir de la forma más transparente y rápida posible entre una serie de especialistas y servicios de emergencia, que no tiene sentido elegir otra opción que no sea la de implementar una infraestructura de tiempo real la cual soporte comunicación dúplex completa.

4.2.1. Tecnologías de comunicación en tiempo real

Una vez ha quedado patente que la comunicación full-duplex en tiempo real es la más apropiada para el proyecto que se trata en este documento es necesario plantear el siguiente paso: determinar cuál de todas las tecnologías disponibles en la actualidad será con la que se implemente el sistema.

Existen múltiples tecnologías de comunicación en tiempo real (RTC: *real-time communication*) en el mercado, por lo que para esta exposición se han seleccionado algunas de las más importantes en la actualidad, las cuales se pretenden introducir de forma breve a lo largo de este apartado pasando finalmente a realizar un desglose de las características más importantes de las mismas en forma de tabla.

4.2.1.1. HTTP Live Streaming

HTTP Live Streaming, o HLS, es un protocolo de comunicaciones con soporte para bitrate variable desarrollado por Apple con el fin de permitir a los proveedores de contenido enviar video y audio (ya sea en vivo o pregrabado) a los dispositivos iPhone o iPod usando servidores web convencionales.[?]

Con el fin de lograr una amplia aceptación y estandarización de HLS, Apple ofrece las especificaciones del mismo tanto en la web como en un borrador IETF.[?] Este objetivo se ha conseguido pues según una encuesta realizada en 2019 por Bitmovin HLS tiene gran peso en la industria y es usado por un gran porcentaje de proveedores de *streaming*, concretamente el 79 %.[?]

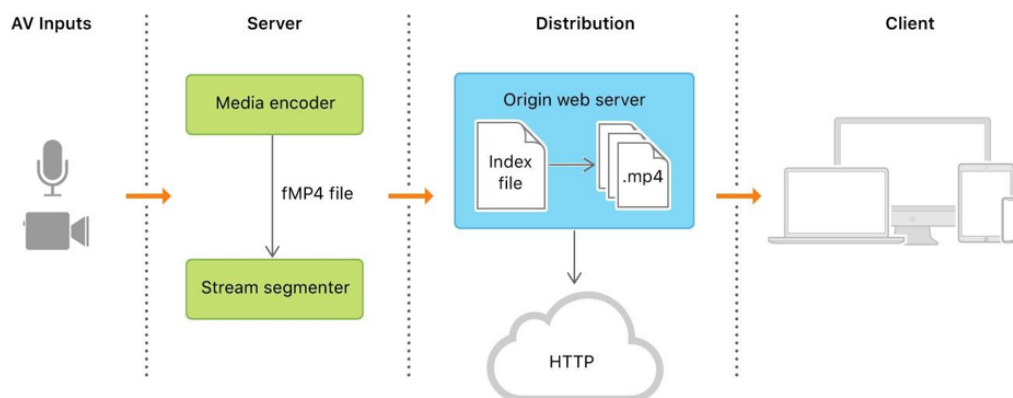


Figura 4.9: Componentes de HTTP Live Streaming [?]

El funcionamiento de HLS se basa en transacciones HTTP estándar: un flujo de datos se codifica en diferentes tasas de bits para ser después segmentado en diversas secciones de igual duración. Un cliente puede a continuación solicitar estas porciones de vídeo a través de una conexión HTTP y eligiendo el *bitrate* más adecuado en cualquier momento dependiendo de las condiciones de la red.[?]

Una de las ventajas que trae consigo el uso de HTTP es poder pasar a través de cualquier firewall o proxy que tenga habilitado este tipo de tráfico, a diferencia de los protocolos basados en UDP (como RTP y WebRTC), por lo que es posible ofrecer contenido desde servidores y CDN convencionales. Además, soporta encriptación de datos AES y HTTPS, lo cual se puede usar a modo de DRM de ser requerido.[?]

4.2.1.2. Web Real-Time Communication

Web Real-Time Communication (WebRTC por sus siglas en inglés) es una tecnología HTML5 [?] inicialmente desarrollada por la empresa Global IP Solutions (GIPS) y posteriormente adquirida por Google, la cual cuenta con la colaboración de otros grandes participantes del mercado de los navegadores web (principalmente Opera y Mozilla).[?]

WebRTC nace con el objetivo de integrar herramientas nativas en los navegadores para hacer posible la integración sencilla de servicios de intercambio de datos en tiempo real en aplicaciones web. Estos flujos de información pueden incluir audio y vídeo de los participantes, captura de pantalla, archivos de todo tipo, mensajería de texto, etc.[?]

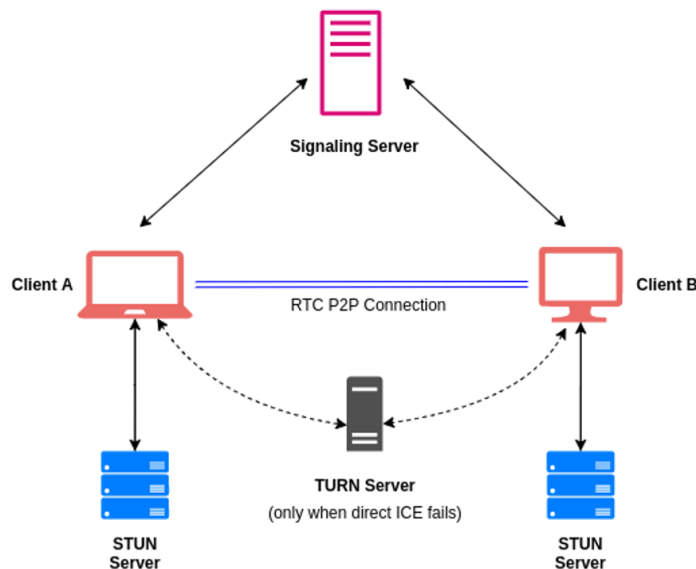


Figura 4.10: Esquema de comunicación mediante WebRTC [?]

Una ventaja importante de WebRTC es su baja latencia pues está basado en UDP y, si bien no garantiza de ninguna manera que los paquetes de datos lleguen correctamente a destino, gracias a esto obtiene mejoras en cuanto al retraso en aplicaciones en las que la latencia es más importante que la calidad de imagen. Además, permite a los participantes de una conferencia estar conectados entre sí de forma directa de una manera *peer-to-peer* (P2P), usando servidores costosos solamente para las negociaciones necesarias para el establecimiento y el mantenimiento de la conexión.[?]

4.2.1.3. Dynamic Adaptive Streaming over HTTP

Dynamic Adaptive Streaming over HTTP (o MPEG-DASH) fue la primera tecnología de transmisión de vídeo en tiempo real con bitrate variable basada en HTTP en convertirse en un estándar internacional (ISO/IEC 23009-1:2012 publicado en abril de 2012) [?]. MPEG-DASH es una tecnología de comunicación RTC similar a la anteriormente mencionada HLS en el sentido de que posibilita la transmisión de *streams* multimedia a través de servidores HTTP convencionales.[?]

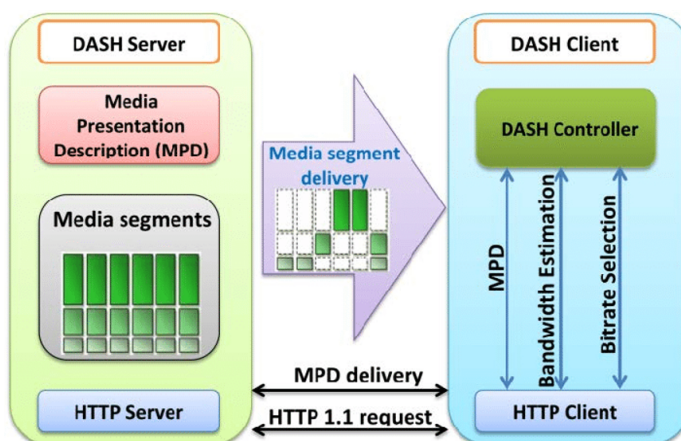


Figura 4.11: Visión general de Dynamic Adaptive Streaming over HTTP [?]

Al igual que HTTP Live Streaming, funciona dividiendo los flujos de datos en pequeños segmentos codificados a diferentes tasas de bits que serán unidos a un archivo *manifest* en el servidor. Este archivo manifest incluye una dirección URL para cada uno de los segmentos codificados en cada una de las tasas de bits posibles.

A continuación, los clientes deberán elegir y descargar los fragmentos que contengan la mayor calidad de imagen y sonido posible sin rebasar la capacidad de la CPU del dispositivo en el que se encuentran o el ancho de banda disponible con el fin de evitar paradas en el stream.[?] Una ventaja de MPEG-DASH es su agnosticismo en cuanto a códecs,[?] por lo que es decisión de los desarrolladores integrar H.264, VP8, VP9, etc, según sus necesidades.

4.2.1.4. Comparación de tecnologías

Después de esta breve introducción de tecnologías, se ofrece una tabla resumen de las características más importantes de las mismas en la que se pueden apreciar propiedades como la latencia, soporte para DRM, su escalabilidad, etc:

Tabla 4.1: Comparación de tecnologías de comunicación en tiempo real

	HLS	WebRTC	MPEG-DASH
Latencia	Mayor de 5 segundos. Unos 2 segundos para el caso de LHLS.	Menor de 1 segundo.	Mayor de 3 segundos.
Soporte para DRM	Sí.	Sí.	Sí.
Escalabilidad	Alta.	Baja.	Alta.
Bitrate adaptativo	Sí.	Sí.	Sí.
Soporte para encriptación	Sí.	Sí.	Sí.
Facilidad para atravesar NAT y firewalls	Alta.	Media.	Alta.
Redundancia	Media.	Baja.	Media.
Códecs soportados	H.264, H.265.	H.264, VP8, VP9.	Agnóstico.
Ejemplos comerciales	Apple TV.	Google Hangouts, YouTube Live.	YouTube, Netflix.
Herramientas específicas para HoloLens 2	No.	Sí, MixedReality WebRTC.	No.

4.2.2. ¿Por qué WebRTC?

En el pasado la comunicación en tiempo real requería de grandes y costosos recursos tecnológicos,[?] la instalación de *plugins* (complementos) por parte de los usuarios y existía gran complejidad técnica a la hora de integrar en aplicaciones preexistentes, haciendo difícil la popularización de estos útiles servicios que idealmente deberían ser tan transparentes y sencillos para el cliente como el uso de una hoja de cálculo o un formulario web.

Google contaba con gran experiencia en la comunicación RTC principalmente debido al desarrollo de sus productos Gmail Video Chat (2008) y Hangouts (2011). En mayo de 2010 adquirieron Global IP Solutions (GIPS), quienes habían desarrollado una gran cantidad de tecnologías requeridas para el buen funcionamiento de las comunicaciones RTC, por ejemplo, sistemas de cancelación de eco y algunos códecs de audio y vídeo. Google posteriormente convirtió en open source todos los componentes de WebRTC y comenzaron una colaboración con IETF y W3C para lograr convertirlo en un estándar de la industria.[?]

WebRTC se ha convertido en la tecnología de comunicación en tiempo real utilizada en este proyecto debido a una serie de ventajas: baja latencia, estandarización, soporte multiplataforma y al hecho de ya existir una biblioteca desarrollada por Microsoft con el fin de su uso en el desarrollo de aplicaciones para el *headset* HoloLens 2.

Al estar basado en HTML5, WebRTC es soportado por una gran cantidad de dispositivos y navegadores web, lo que lo hace un candidato ideal y abre la puerta de cara al futuro a añadir nuevas plataformas a la lista de las soportadas por el sistema. Además está estandarizado evitando problemas de compatibilidad entre plataformas y, al ser de código abierto, está sometido a un flujo constante de mejoras y correcciones por parte de una comunidad de desarrolladores distribuidos a lo largo de todo el mundo.

La característica más importante de cara a este desarrollo es su baja latencia, pues es vital mantener la comunicación lo más fluida posible por dos razones: en una situación como son las que está orientado este proyecto, la diferencia entre el éxito y el fracaso de la actuación de los servicios de emergencia puede ser de décimas de segundo y, además una alta latencia podría dificultar la comunicación en tiempo real, convirtiendo lo que debería ser una experiencia transparente y enriquecedora para el usuario en una distracción peligrosa.

Finalmente, la existencia de herramientas y bibliotecas desarrolladas por Microsoft para su uso en HoloLens 2 (MixedReality WebRTC) garantiza la viabilidad del proyecto y la compatibilidad de la tecnología con el headset a la vez que proporciona un buen ahorro en costes y tiempo.

Por supuesto es cierto que WebRTC tiene algunas desventajas, principalmente relativas a su naturaleza *peer-to-peer*: la escalabilidad es reducida debido a la necesidad de crear un enlace P2P con cada participante en la conversación, por lo que para compartir un vídeo de 1 Mbps de bitrate con 10 personas se acabará necesitando 10 Mbps de ancho de banda.

Aún así, WebRTC funciona bien para grupos pequeños como los que vamos a tratar a lo largo del desarrollo de este proyecto (que normalmente estarán formados únicamente por una persona desplegada en la localización de un accidente y el especialista apropiado), pero dejando abierta además la posibilidad abierta a que se una un número reducido de participantes más si en el futuro se desea implementar esta funcionalidad.

CAPÍTULO 5

Resultados

En el presente capítulo se trata el sistema finalizado y la arquitectura que se ha desarrollado a lo largo de los meses que ha durado la ejecución del proyecto. Para mantener el documento lo más breve posible se ha decidido hablar únicamente de las partes más relevantes de su diseño e implementación.

5.1. CONSIDERACIONES PREVIAS

El sistema (también conocido como *Health-5G*) se compone de cuatro aplicaciones con responsabilidades diferenciadas divididas en dos temáticas diferentes: por un lado están las aplicaciones de videoconferencia y por otro las que se encargan de dar soporte a la integración con dispositivos médicos.

En cuanto a las aplicaciones relativas a la **videoconferencia** encontramos los siguientes desarrollos:

- **Health-5G_Holo**: contiene la solución destinada a ejecutarse en el *headset* HoloLens 2 por parte del equipo de emergencias.
- **Health-5G_Desktop**: aplicación para el computador personal del médico especialista.

A continuación, se exponen las aplicaciones relativas a la **integración con dispositivos médicos**:

- **Health-5G_Client**: cliente de captura de señal de vídeo proveniente de dispositivos médicos. Se ejecutará en un PC cercano al equipo de emergencias.
- **Health-5G_Server**: servidor en la nube con el fin de distribuir las imágenes obtenidas por *Health-5G_Client* a las aplicaciones de videoconferencia.

En la figura ?? se expone la arquitectura del sistema, a muy alto nivel, para ofrecer una visión general del mismo:

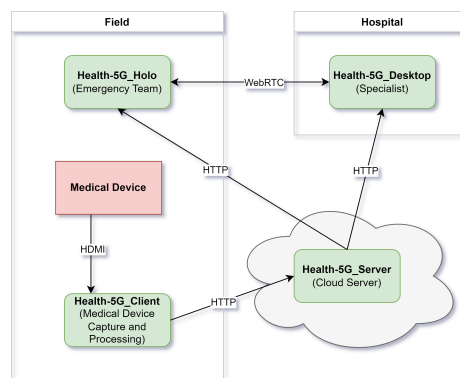


Figura 5.1: Diagrama de comunicación entre aplicaciones. Las flechas representan el flujo de la información.

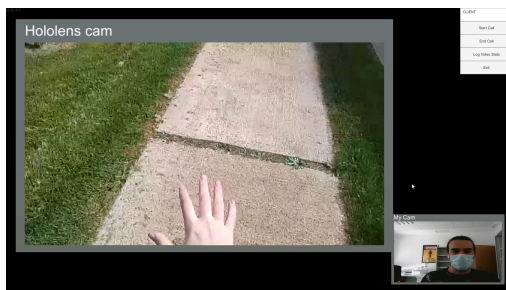
5.2. RESULTADOS OBTENIDOS

A lo largo de este apartado se hablará en detalle de los resultados obtenidos al acabar proyecto desde la perspectiva del usuario final, dejando los detalles técnicos para más adelante. Se pretende con esto proporcionar al lector una visión general del sistema finalizado.

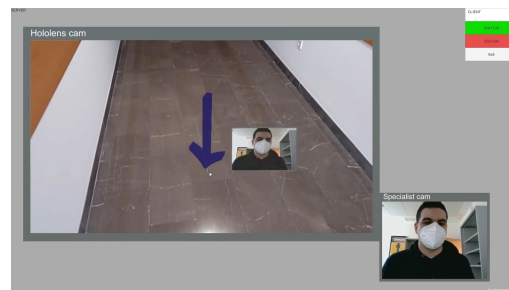
Una parte muy importante de este trabajo fin de máster ha consistido en el desarrollo de unos sistemas de interacción ágiles y eficientes con el fin de permitir una interacción cómoda para los usuarios. Es imperativo tener en cuenta que es un conjunto de aplicaciones diseñado para ser usado en situaciones de emergencia donde cada segundo puede ser la diferencia entre el éxito y el fracaso.

Las principales condiciones que se han intentado solventar a la hora de realizar el diseño de la interfaz han sido dos: la primera es que la experiencia no debe ser intrusiva para el usuario del *headset* y la segunda es que debe permitir poner marcadores en el espacio 3D a un médico especialista que usa la aplicación en un ordenador con pantalla tradicional en dos dimensiones.

Satisfacer estas condiciones no ha sido un proceso sencillo ni rápido, y se ha producido una evolución constante durante todo el desarrollo. A modo de referencia se incluyen en la figura ?? algunas capturas de pantalla obtenidas en diferentes puntos en el tiempo.



(a) Primera versión, solo videoconferencia



(b) Introducción de los marcadores en 3D



(c) Nuevas herramientas y formas de interacción

Figura 5.2: Evolución desde la vista de la aplicación *Health-5G_Desktop*

De vuelta al presente, se puede ver en la imagen ?? que sigue a este párrafo la perspectiva del usuario de la aplicación de PC (la cual incluye el punto de vista del casco de realidad aumentada) en la versión final:



Figura 5.3: Vista desde la aplicación del especialista durante una videoconferencia

La interfaz de usuario para la aplicación utilizada por el médico especialista está basada en el clásico paradigma de escritorio con una pequeña particularidad: se ha debido encontrar una manera de colocar marcadores en el espacio 3D que ocupa el *headset* perteneciente al equipo de emergencias a través de una pantalla en dos dimensiones. En la captura ?? se pueden ver resaltados los diferentes componentes de la interfaz, que serán detallados más adelante.

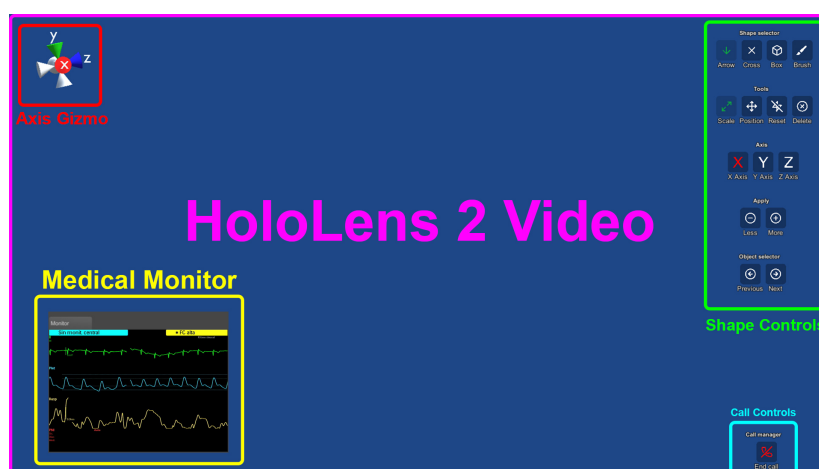


Figura 5.4: Desglose de la aplicación para PC





El componente que ocupa todo el fondo de la aplicación es el vídeo proveniente de la cámara de HoloLens 2 (indicado como **HoloLens 2 Video** en el diagrama) y es de especial relevancia por dos motivos: muestra el punto de vista del equipo de emergencias y es la puerta de entrada a la interacción por parte del médico especialista pues permite el emplazamiento de marcadores en el espacio tridimensional mediante clics. Debido a su importancia y tras varias iteraciones se ha determinado que maximizar el tamaño del mismo, por lo que se extiende por toda la aplicación y no está enmarcado de ninguna manera.

En la esquina superior izquierda se encuentra un indicador de orientación (**Axis Gizmo**) que muestra la dirección en la que mira el usuario del *headset*. Si bien puede parecer algo trivial, este objeto surge de la necesidad de mostrar de alguna manera al médico especialista la orientación en la que mira el personal del equipo de emergencias para facilitar el escalado y reposición de los diferentes marcadores.

Los botones con los que cuenta la aplicación están agrupados en controles de forma (**Shape Controls**) y controles de llamada (**Call Controls**) y serán explicados a continuación agrupados en base a la temática de las funciones que realizan.





Los botones de selección de forma permiten elegir el tipo de marcador que se pondrá en el espacio tridimensional al hacer clic en el vídeo proveniente del *headset*. El caso del pincel (*brush*, en inglés) es especial ya que no tiene forma predefinida, la establecerá el usuario al dibujar directamente sobre el vídeo con el ratón de su computador.

Tabla 5.1: Botones de selección de forma (**Shape Selector**)

Icono	Nombre	Observaciones
	Arrow	Selecciona el marcador con forma de flecha.
	Cross	Selecciona el marcador con forma de cruz.
	Box	Selecciona el marcador con forma de caja.
	Brush	Selecciona el marcador de dibujo a mano alzada.




El sistema cuenta con cuatro herramientas con las que manipular los marcadores una vez colocados en el espacio. Las dos primeras de la lista siguiente dan la posibilidad de modificar la escala y la posición en cualquier eje. También hay un botón para restaurar la posición y la escala a los valores originales en cualquier momento. Finalmente, el botón de borrar permite eliminar un marcador del espacio cuando ya no es necesario.

Tabla 5.2: Botones de selección de herramienta (**Tools**)

Icono	Nombre	Observaciones
	Scale	Herramienta que permite escalar el marcador seleccionado.
	Position	Herramienta que permite mover el marcador seleccionado.
	Reset	Esta herramienta devuelve el marcador seleccionado a su posición y escala original.
	Delete	La herramienta de borrado elimina el marcador seleccionado.



Las herramientas de escala y posición serán aplicadas en los ejes determinados mediante los siguientes botones. Al pulsar uno de los botones de los ejes será marcado o desmarcado, pudiendo así conseguir cualquier combinación posible.

Tabla 5.3: Botones de selección de eje (*Axis*)

Icono	Nombre	Observaciones
	X Axis	Marca o desmarca el eje X para uso con las herramientas de escala y posición.
	Y Axis	Marca o desmarca el eje Y para uso con las herramientas de escala y posición.
	Z Axis	Marca o desmarca el eje Z para uso con las herramientas de escala y posición.



Una vez seleccionada la herramienta y los ejes se deberá aplicar sobre el marcador correspondiente. Estos botones permiten al usuario disminuir o aumentar la escala y la posición según desee.

Tabla 5.4: Botones de aplicación de la herramienta (*Apply*)

Icono	Nombre	Observaciones
	Less	Disminuye la escala o la posición en los ejes marcados.
	More	Aumenta la escala o la posición en los ejes marcados.




Estos botones alternan la selección entre los marcadores colocados en el espacio. El estado de un marcador se mostrará gráficamente al usuario mediante un cambio de color: rosa si está seleccionado, blanco si no lo está.

Tabla 5.5: Botones de selección de marcador (*Object Selector*)

Icono	Nombre	Observaciones
	Previous	Selecciona el marcador anterior.
	Next	Selecciona el marcador siguiente.

Finalmente el botón de llamada permite iniciar o detener la comunicación. En cualquier momento dado habrá presente solo uno de los botones indicados a continuación:

Tabla 5.6: Botones de control de llamada (*Call Manager*)

Icono	Nombre	Observaciones
	Call	Permite iniciar la llamada.
	Calling...	La llamada está en proceso de inicio. En este estado el botón no es interactivo.
	End call	Si se pulsa, termina la llamada.

El ultimo componente de la interfaz de la aplicación del especialista es el monitor de constantes vitales (*Medical Monitor*). Este objeto consiste en una ventana de forma y tamaño modificable en el que el especialista tendrá acceso a los datos médicos en tiempo real de la persona a la que se está atendiendo a distancia.

A continuación se tratará la aplicación orientada al equipo de emergencias. La particularidad de este desarrollo es lo novedoso del paradigma de realidad mixta que implementa que además debe proveer una funcionalidad lo más sencilla y no intrusiva posible para permitir al profesional sanitario realizar su trabajo de la forma más adecuada. En la imagen ?? se incluye una captura de pantalla de esta aplicación resaltando los componentes principales.

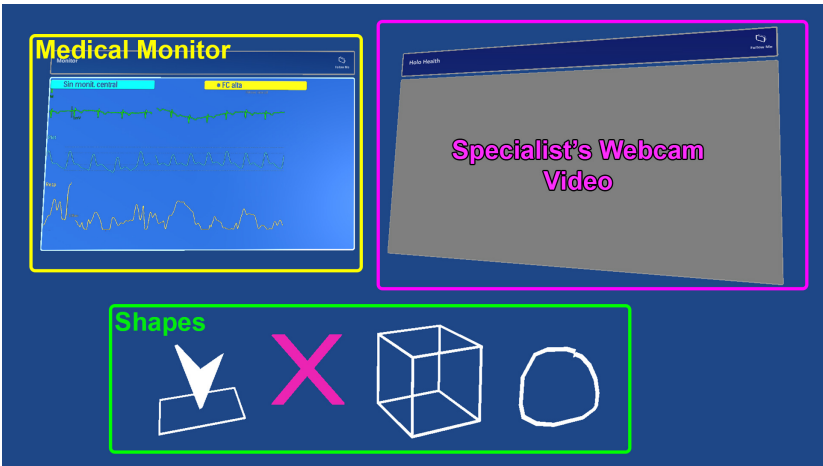
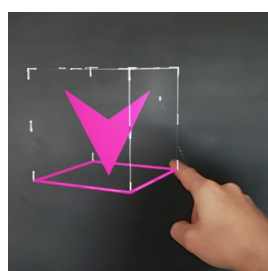


Figura 5.5: Desglose de la aplicación de HoloLens 2

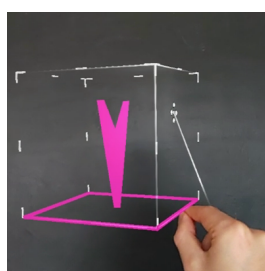
En la figura ?? se pueden ver las diferentes formas (*Shapes*) con las que cuenta la aplicación, las cuales hacen la función de marcadores y serán presentadas en color rosa cuando están seleccionadas y en blanco en caso contrario. Esta selección de colores no es trivial pues a lo largo de las pruebas se ha determinado empíricamente que estos tonos son muy visibles para el usuario de HoloLens 2. A continuación se ofrece una pequeña descripción de cada una de estas formas:

- **Arrow** (flecha): flecha útil para marcar de forma visible puntos concretos en el espacio.
- **Cross** (cruz): cruz que siempre mira directamente a cámara, tiene el rol de marcador secundario.
- **Box** (caja): caja con sólo las aristas visibles. Pensada para poder envolver completamente objetos.
- **Line** (dibujo a mano alzada): representa una línea dibujada a mano alzada por el usuario de PC y puede adoptar cualquier forma.

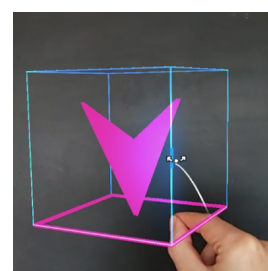
Todos estos objetos por supuesto se pueden escalar, mover y borrar desde la aplicación del médico especialista usando los mecanismos de interacción descritos anteriormente. Además, el usuario de HoloLens 2 podrá mover libremente estos marcadores en el espacio tridimensional en base a los gestos que se pueden ver en la figura ??.



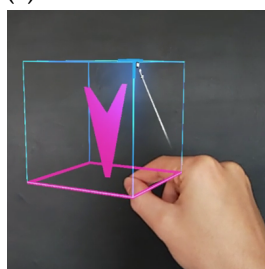
(a) Enfocando un marcador



(b) Moviendo un marcador



(c) Rotando un marcador



(d) Escalando un marcador

Figura 5.6: Usuario interactuando con un marcador holográfico

Cuando un marcador esté enfocado se podrá usar sobre este una serie de comandos de voz:

Tabla 5.7: Comandos de voz de los marcadores holográficos

Comando (español)	Comando (inglés)	Función
Borrar	Remove	Elimina el marcador al que se está mirando.
Reset	Reset	Devuelve el marcador enfocado a su posición, escala y rotación original.
Holo	Holo	Selecciona el marcador enfocado.

Health-5G_Holo cuenta con dos ventanas en las que el usuario verá las constantes vitales del paciente y la cámara web del ordenador del especialista (*Medical Monitor* y *Specialist's Webcam Video*, respectivamente). La particularidad de estas ventanas es que se pueden mover libremente por el escenario mediante gestos, así como cambiar su tamaño, ocultarlas, rotarlas y hacer que nos sigan. Las ventanas holográficas están estructuradas de la siguiente manera:

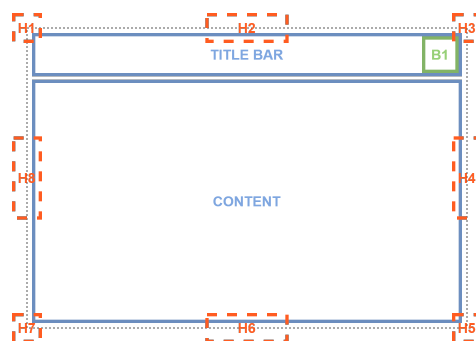
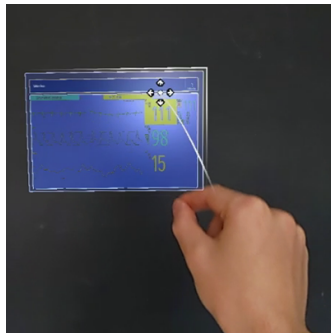
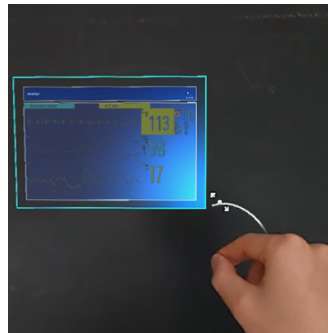


Figura 5.7: Esquema de una ventana holográfica

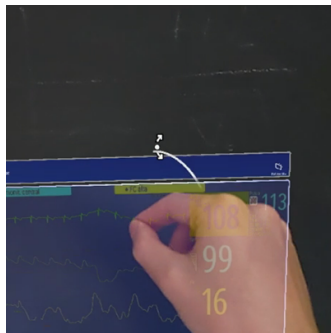
- **Title Bar** (barra de título): esta barra tiene una triple función, por un lado muestra el título de la ventana pero también tiene la responsabilidad de contener algunos botones y servirá como punto de referencia a la hora de reposicionarla mediante gestos.
- **B1** (botón "sígueme"): incluido dentro de la barra del título, permite indicar a esa ventana de manera individual que debe seguir o no al usuario de la aplicación.
- **Content** (contenido): superficie útil de la ventana. Incluye toda la información que es necesario que el usuario tenga a mano.
- **H1-H8** (asas): sirven para rotar o escalar la ventana, según el gesto con el que se manipulen.



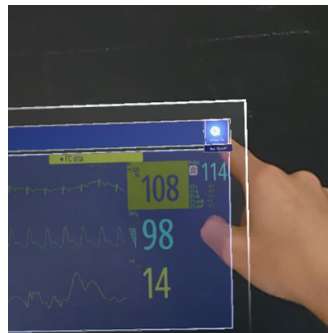
(a) Reposicionamiento de la ventana



(b) Cambio de tamaño de la ventana



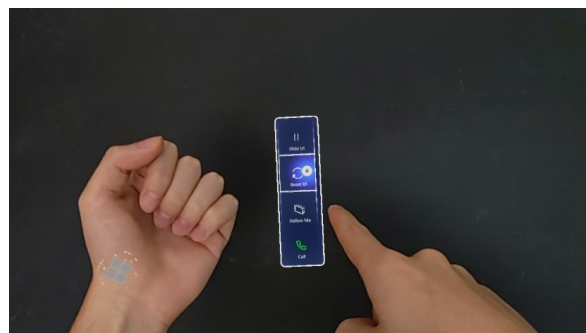
(c) Rotación de la ventana



(d) Cambio de modo de seguimiento

Figura 5.8: Usuario interactuando con una ventana holográfica

Otro componente interactivo de la aplicación es el **menú mano**. El menú mano ofrece una serie de botones útiles y aparecerá cuando el usuario ponga la palma de la mano mirando hacia si mismo.

**Figura 5.9:** Menú mano

Los botones con los que cuenta este menú son los siguientes:

- **Hide UI/Show UI**: muestra u oculta las ventanas de la aplicación.
- **Reset UI**: reposiciona ambas ventanas de la aplicación. Útil si han quedado muy lejos del usuario o en una posición incomoda.
- **Follow Me**: hace que las ventanas de la aplicación sigan o dejen de seguir al usuario del *headset*.
- **Call**: inicia una llamada con el médico especialista.

Finalmente, el usuario podrá utilizar una serie de comandos de voz en cualquier momento que no requieren que esté enfocado ningún holograma concreto. Estos comandos son:

Tabla 5.8: Comandos de voz generales

Comando (español)	Comando (inglés)	Función
Mostrar	Show	Muestra las ventanas de interfaz si se encuentran ocultas.
Ocultar	Hide	Ocultas las ventanas de la interfaz.
Sígueme	Follow	Hace que las ventanas de la aplicación sigan al usuario.
Stop	Stop	Con este comando las ventanas de la aplicación dejarán de seguir al usuario.
Centrar	Center	Reposiciona las ventanas de la aplicación.

5.3. ARQUITECTURA DEL SISTEMA

Visto el proyecto desde la perspectiva del usuario es momento de entrar en la parte técnica del mismo. La arquitectura del sistema se puede visualizar como una aplicación distribuida donde unas partes de esta se ejecutan en el lugar de una emergencia (principalmente en una ambulancia), otras en el hospital en el que se encuentran los médicos especialistas y otras en la nube. El diagrama ?? muestra las partes más importantes del sistema, así como su ubicación física habitual.

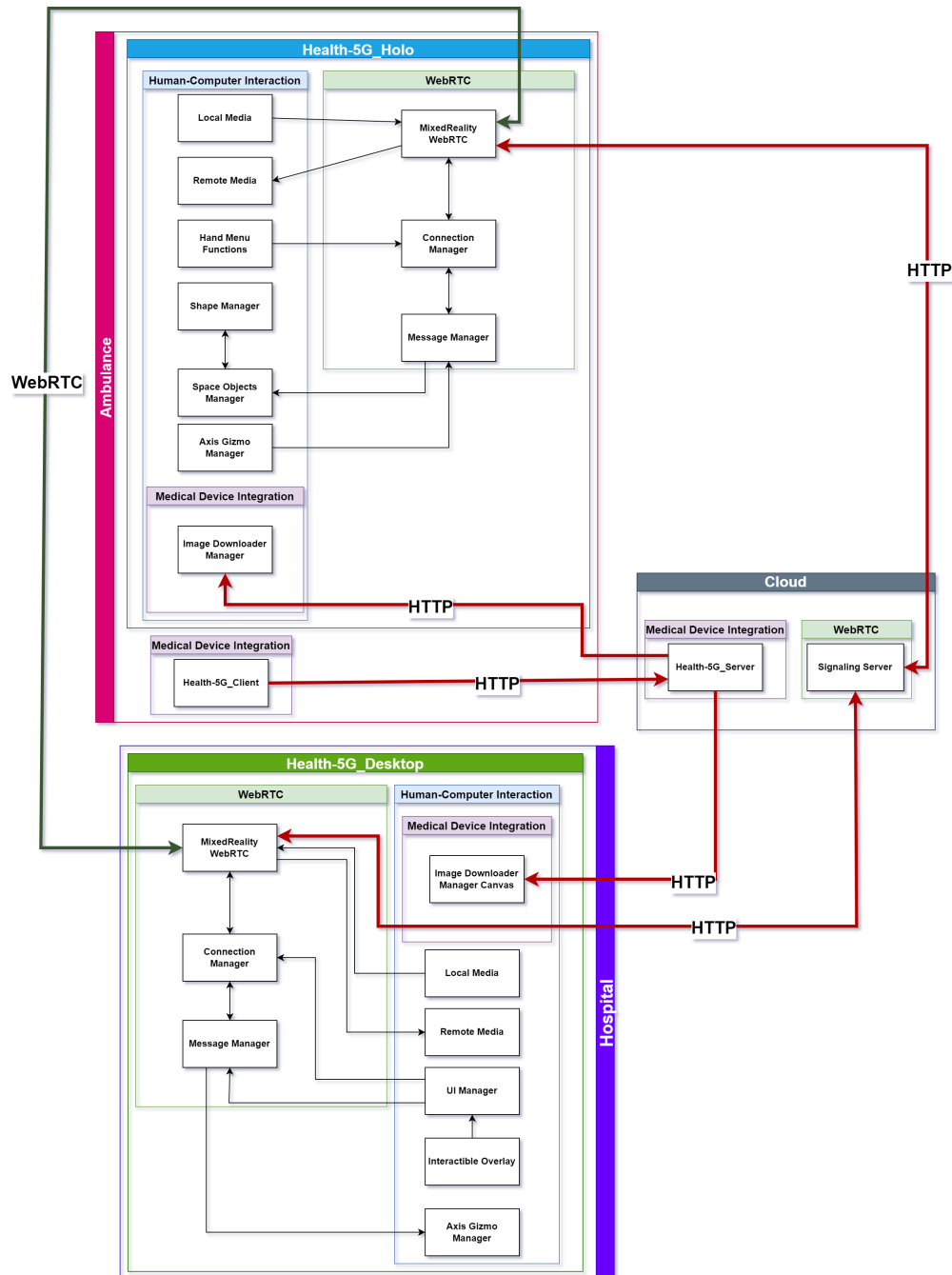


Figura 5.10: Arquitectura del sistema. Las flechas representan la dirección del intercambio de información.

El sistema está dividido en tres grandes subsistemas, cuyos detalles técnicos se exponen a continuación:

1. **Integración con dispositivos médicos:** incluye el proceso de captura y distribución de imágenes procedentes de los dispositivos médicos con los que cuenta la ambulancia.
2. **Comunicación bidireccional WebRTC:** comunicación de baja latencia de audio, vídeo y datos entre las aplicaciones de videoconferencia.
3. **Interacción persona-ordenador:** sistemas que proveen la interacción con el usuario.

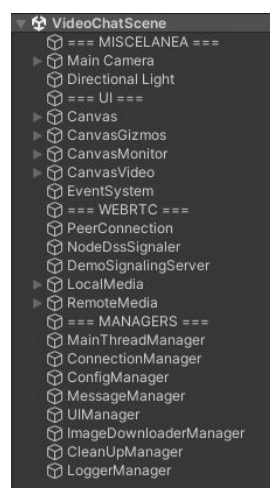
El orden de los bloques no es casual: el subsistema de integración con dispositivos médicos no tiene dependencias con los otros dos, por lo que se verá en primer lugar. Por otro lado, antes de indagar en el módulo de interacción persona-ordenador es necesario saber cómo funciona la comunicación WebRTC (introducida en el punto ??).

Se pueden ver en la figura ?? las aplicaciones que se dedican a facilitar la integración con dispositivos médicos. Estos dos programas se han escrito en el lenguaje python y se comunican entre sí y con las aplicaciones de videoconferencia utilizando el protocolo HTTP.

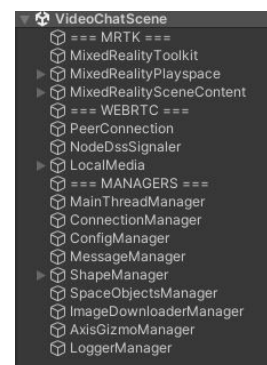
El objetivo más importante de este proyecto es conseguir una experiencia de comunicación bidireccional estable, fluida y de baja latencia entre dos dispositivos de características muy diferentes, en este caso el visor de realidad mixta HoloLens 2 y un computador de escritorio. Con el fin de satisfacer este requisito se ha desarrollado una arquitectura compuesta por dos programas que se comunican entre sí a través de WebRTC.

Estas aplicaciones se han escrito en el lenguaje de programación C# sobre el motor de desarrollo de videojuegos Unity, en concreto la versión 2021.3.1f1, por la facilidad que este provee a la hora de implementar aplicaciones para realidad mixta y por la experiencia previa con la que ya se contaba de proyectos anteriores tales como *RoboTIC: un juego serio basado en realidad aumentada para el aprendizaje de la programación*.¹

Unity usa un sistema basado en escenas. Estas escenas son estructuras de datos que incluyen los objetos (ya sean los scripts que contienen la lógica, archivos de sonido, texturas, modelos, etc) que forman la aplicación que se está desarrollando. En este caso se ha decidido que cada una de las aplicaciones de videoconferencia esté compuesta de una sola escena (VideoChatScene.unity) organizando por temáticas los objetos que la forman, lo que se puede ver claramente en la imagen ??.



(a) Escena de la aplicación de PC



(b) Escena de la aplicación de HoloLens 2

Figura 5.11: Escenas de Unity mostrando los componentes principales

¹<https://github.com/OhEsPaco/RoboTIC>

5.3.1. Integración con dispositivos médicos

En esta sección se trata la integración con dispositivos médicos, en concreto se ha usado para las pruebas el monitor **Philips MX450** del que se pueden obtener datos de frecuencia cardiaca, presión arterial, saturación de oxígeno en sangre y electrocardiograma.



Figura 5.12: Monitor médico usado en el proyecto

Con el fin de permitir la integración de cualquier otro aparato médico de forma sencilla, y debido a la dificultad de obtener acceso a los datos de la mayoría de dispositivos médicos comerciales, se ha apostado por un enfoque genérico en el que los datos se obtienen a través de la señal de vídeo procedente del dispositivo. En el sistema actual la imagen es procesada en un computador cercano al aparato médico para después ser enviada a la nube con el fin de ser distribuida para su uso por parte de las aplicaciones de videoconferencia. En la imagen ?? aparecen los componentes de la arquitectura de esta parte del sistema.

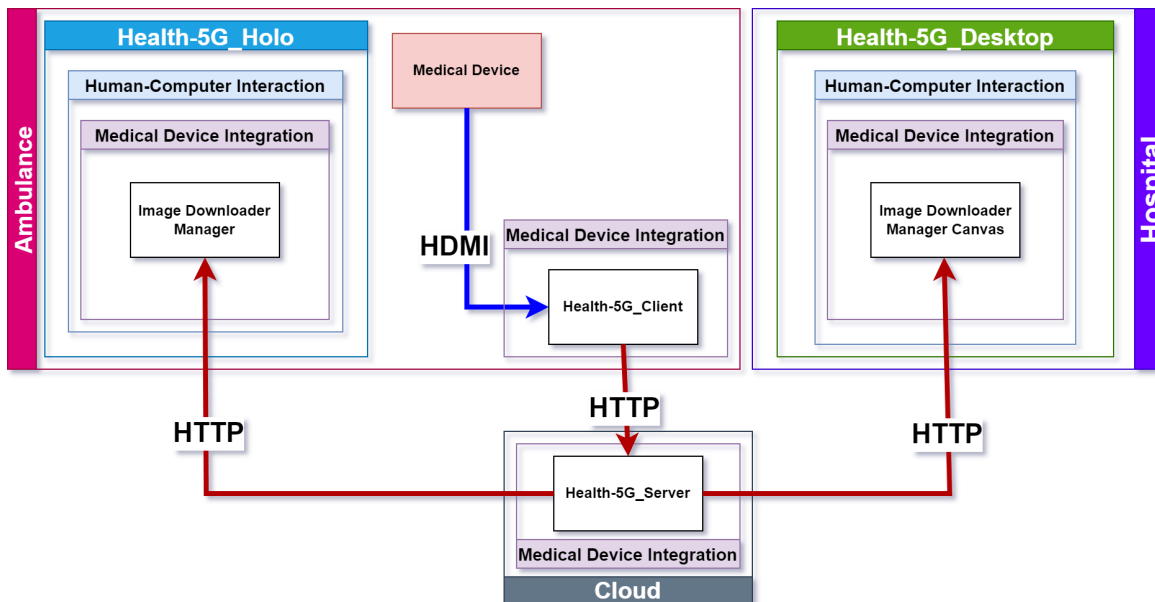


Figura 5.13: Sistema de integración con dispositivos médicos. Las flechas representan la dirección del intercambio de información.

Este subsistema se dividirá en tres partes: las que funcionan en la ambulancia (Health-5G_Client, Image Downloader Manager), en el hospital (Image Downloader Manager Canvas) y en la nube (Health-5G_Server). Para realizar la explicación técnica se seguirá el orden en el que viaja la información, por lo que se empezará por Health-5G_Client.

Health-5G_Client (o cliente) es una pieza fundamental en la integración con dispositivos médicos pues su responsabilidad principal es la de capturar la señal de vídeo de estos aparatos para posteriormente enviarla a la nube. Este cliente está compuesto de un sólo *script* (webclient.py) escrito en el lenguaje de programación python. En el listado de código ?? se muestra la parte más importante del *script*: un bucle infinito de captura, codificación y envío de imágenes.

Listado 5.1: Bucle de envío de imágenes

```
1 cam = cv2.VideoCapture(camera)
2
3 while True:
4     ret, frame = cam.read()
5     if not ret:
6         continue
7
8     (flag, encodedImage) = cv2.imencode('.jpg', frame)
9     if not flag:
10        continue
11
12    data = pickle.dumps(encodedImage, 5)
13    requests.post(url, files={'media': data},
14                  headers={'api-key': apiKey})
```

La ejecución comienza creando un objeto *VideoCapture* de la biblioteca *opencv-python*², pasándole como argumento el número de la capturadora HDMI a la cual está conectada el dispositivo médico (o un nombre de archivo, en caso de que se quiera transmitir un vídeo).

A continuación empieza un bucle en el que el primer paso será intentar leer una imagen desde la capturadora que será codificada en formato JPG tras comprobar que se ha capturado correctamente. La codificación del cuadro en este formato no es casual y se ha elegido tras diversas pruebas pues proporciona una calidad de imagen más que suficiente sin requerir tanto ancho de banda como PNG o el envío de los datos en crudo, lo que llegaba a saturar la conexión entre cliente y servidor. Una vez realizada la codificación con éxito llega el momento de serializar la imagen para poder enviarla a través de la red, para esto se usa la biblioteca de serialización de objetos de python, *pickle*.

Finalmente, se hace una petición POST a la dirección del servidor donde se encuentre alojado el programa Health-5G_Server. A modo de medida de seguridad, y para evitar que cualquier persona pueda subir imágenes, se define en la cabecera de la petición una clave de API para asegurar el uso autorizado del servicio.

La aplicación **Health-5G_Server** es la parte en la nube del sistema de integración de dispositivos médicos y expone dos *endpoints*: uno para subir imágenes y otro para descargarlas. Programada en python utilizando el framework Flask³, está completamente *dockerizada* y preparada para la integración continua utilizando *pipelines* en Bitbucket.

El *endpoint* para subir imágenes (ubicado en la ruta /) sirve para atender las peticiones POST que hace el cliente de captura de imágenes anteriormente explicado. El código que da soporte a este procedimiento es el siguiente:

²<https://pypi.org/project/opencv-python/>

³<https://flask.palletsprojects.com/>

Listado 5.2: Endpoint de recepción de imágenes

```

1 @app.route('/', methods=['POST'])
2 def uploadImage():
3     global outputFrame, lock, width, height, apiKey
4     if 'media' not in request.files:
5         return 'there is no media'
6     if 'api-key' not in request.headers:
7         abort(403)
8
9     requestApiKey = request.headers['api-key']
10    if requestApiKey is None or requestApiKey != apiKey:
11        abort(403)
12
13    data = request.files['media'].read()
14    frame = pickle.loads(data, encoding="bytes")
15    frame = cv2.imdecode(frame, cv2.IMREAD_COLOR)
16    with lock:
17        width = frame.shape[1]
18        height = frame.shape[0]
19        outputFrame = frame
20    return 'ok'

```

En la primera parte simplemente se comprueba que la petición contenga una imagen y que la clave de la API que envía el cliente concuerde con la clave que tiene guardada el servidor. Seguidamente se continúa leyendo el archivo subido para después proceder a su deserialización utilizando la biblioteca pickle y a su decodificación mediante opencv-python. Si todo ha salido bien ahora tendremos en la nube una copia muy similar de la imagen obtenida por la capturadora HDMI del cliente.

El otro *endpoint* del que dispone el servidor es el del que se podrán descargar los clientes las imágenes subidas. El código del mismo se muestra en el listado ??.

Listado 5.3: Endpoint de descarga de imágenes

```

1 @app.route(
2     "/<int:referenceResolutionX>/<int:referenceResolutionY>/
3     <int:cropStartX>/<int:cropEndX>/<int:cropStartY>/<int:cropEndY>"
4 )
5 def downloadImage(
6     referenceResolutionX,
7     referenceResolutionY,
8     cropStartX,
9     cropEndX,
10    cropStartY,
11    cropEndY,
12 ):
13     global apiKey, outputFrame, lock, width, height
14     if outputFrame is None:
15         return 'There is no frame to download'
16     if 'api-key' not in request.headers:
17         abort(403)
18     requestApiKey = request.headers['api-key']
19     if requestApiKey is None or requestApiKey != apiKey:
20         abort(403)
21
22     cropStartX, cropEndX, cropStartY, cropEndY = ↵
23         ↵ calculateFinalCrop(
24             referenceResolutionX,
25             referenceResolutionY,
26             cropStartX,
27             cropEndX,
28             cropStartY,
29             cropEndY,

```

```

30
31     with lock:
32         (flag, encodedImage) = cv2.imencode(
33             ".jpg", outputFrame[cropStartY:cropEndY, ↵
34                 ↵ cropStartX:cropEndX]
35         )
36         if not flag:
37             return 'Error encoding image'
38
39     response = make_response(bytearray(encodedImage))
40     response.headers.set("Content-Type", "image/jpeg")
41     response.headers.set("Content-Disposition",
42         "attachment", filename="pic.jpg")
43
44     return response

```

Es importante fijarse en la gran cantidad de parámetros que recibe este *endpoint* a través de la URL. Estos parámetros son *referenceResolutionX*, *referenceResolutionY*, *cropStartX*, *cropEndX*, *cropStartY* y *cropEndY* y permiten que los clientes descarguen una imagen recortada directamente desde el servidor con el fin de poder mostrar al usuario solamente las partes relevantes de la imagen capturada.

El procedimiento comienza comprobando que haya una imagen para descargar y que la clave de API esté presente y sea correcta. Una vez hecho esto se calcula el corte final en base a los parámetros antes mencionados, además de comprobaciones de seguridad como que el inicio de un corte no sea mayor que el fin. Finalmente, la imagen recortada se codifica como JPG y se añade a la respuesta para que pueda ser descargada por el cliente.

Antes de entrar en la integración con Unity cabe mencionar que se han preferido las peticiones HTTP como manera principal de descarga de imágenes pues el uso de WebSockets producía importantes problemas en HoloLens 2, principalmente caídas de la aplicación lo cual es inadmisibles en un proyecto orientado a situaciones de emergencia.

La integración con las aplicaciones de videoconferencia está basada en dos objetos: **ImageDownloaderManager.cs** para en HoloLens 2 e **ImageDownloaderManagerCanvas.cs** en PC. Estos componentes tienen diferencias mínimas entre sí por lo que para la explicación nos centraremos en el primero de ellos:

Listado 5.4: Bucle de descarga de imágenes

```

1  while (true)
2  {
3      millisAtStart = DateTime.Now.Ticks / ↵
4          ↵ TimeSpan.TicksPerMillisecond;
5      request = UnityWebRequestTexture.GetTexture(MediaUrl + "/" ↵
6          ↵ + referenceResolutionX + "/" + referenceResolutionY + ↵
7          ↵ "/" + cropStartX + "/" + cropEndX + "/" + cropStartY ↵
8          ↵ + "/" + cropEndY);
9      request.SetRequestHeader("api-key", apiKey);
10     yield return request.SendWebRequest();
11
12     if (request.result == UnityWebRequest.Result.Success)
13     {
14         try
15         {
16             textureMaterial.mainTexture = ↵
17                 ↵ request.downloadHandler.texture;
18         }
19         catch {}
20
21         millisDifference = (DateTime.Now.Ticks / ↵
22             ↵ TimeSpan.TicksPerMillisecond) - millisAtStart;
23         if (millisDifference < maxFrameTime)
24         {

```

```

19         yield return new WaitForSecondsRealtime((maxFrameTime -
           ↳ millisecondsDifference) / 1000);
20     }
21 }
22 Resources.UnloadUnusedAssets();
23 request.Dispose();
24 }

```

Se puede ver que se fundamenta en un bucle infinito que descarga imágenes a un ritmo determinado por el *framerate* máximo deseado para posteriormente renderizarlas sobre una textura perteneciente a un objeto de Unity. Esto trae como resultado poder visualizar la pantalla del dispositivo médico en tiempo real tanto en HoloLens 2 como en PC proporcionando así información vital al equipo de emergencias y al médico especialista.

5.3.2. Comunicación bidireccional WebRTC

Antes de desglosar los componentes usados en la comunicación WebRTC es importante conocer a grandes rasgos como funciona esta tecnología. *Web Real-Time Communication* permite el intercambio de diversos flujos de datos conteniendo estos audio, vídeo y cualquier otro tipo de información de forma P2P con la gran ventaja de no requerir ningún tipo de *plugin* ni extensión pues todo lo necesario está integrado de forma nativa en los navegadores más populares como Google Chrome, Mozilla Firefox, Opera, etc.

Pese a basarse en un paradigma *peer-to-peer*, WebRTC también necesita un servidor de señalización (*signaling*, en inglés) para poder operar y curiosamente este no está definido en los estándares con el fin de mantener la compatibilidad con tecnologías ya establecidas y evitar redundancias en la medida de lo posible.[?]

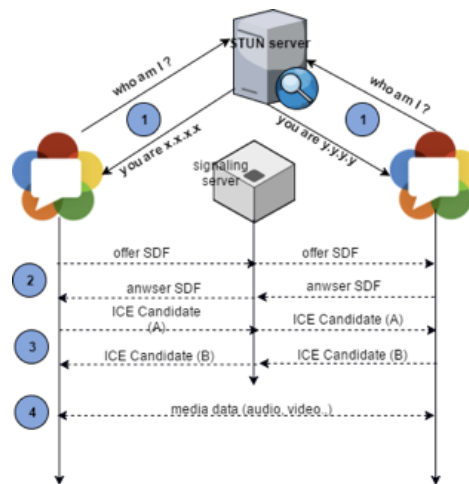


Figura 5.14: Secuencia de señalización de WebRTC[?]

A la hora de realizar una comunicación, los diversos clientes se comunicarán primero con un servidor STUN (Session Traversal Utilities NAT), cuyo cometido es el de proporcionar la dirección IP pública a la máquina que ha hecho la petición correspondiente, pues es necesario conocer esta información para poder iniciar una comunicación P2P. Cada una de las direcciones recibidas de parte de un servidor STUN son llamadas candidatos ICE (Interactive Connectivity Establishment).

A veces es imposible conectar a los clientes de forma directa, por lo que un servidor TURN (Traversal Using Relay NAT) entra en juego actuando a modo de relé recibiendo los mensajes de cada uno de los clientes y reenviándolos al resto, posibilitando la comunicación aún cuando STUN falle y actuando a modo de backup.

Seguidamente los ICE candidates se compartirán entre clientes mediante el servidor de signaling pudiendo por fin establecer la comunicación.[?] A través de este servicio también se llevará a cabo un intercambio de mensajes SDP (Session Description Protocol), que es un protocolo usado con el fin de determinar qué códecs multimedia soporta cada uno de los participantes facilitando el llegar a un acuerdo y la determinación de una tecnología compatible. Un ejemplo breve de esto último sería: si uno de los pares involucrados soporta H.264 y VP8 pero otro sólo soporta VP8 se utilizará este último códec a lo largo de la videoconferencia.

A la hora de transmitir datos en tiempo real, RTP (Real-time Transport Protocol) es usado como el protocolo de transporte de multimedia para WebRTC.[?] Audio y vídeo son transmitidos usando este protocolo sobre UDP y si bien no garantiza que los paquetes de datos lleguen correctamente a su destino obtiene muy bajas latencias debido a la ausencia de comprobaciones.

5.3.2.1. WebRTC en Unity

Para realizar la integración de Unity con WebRTC se ha optado por usar MixedReality WebRTC⁴. Esta biblioteca es una colección de componentes desarrollado por Microsoft con el fin de integrar vídeo y audio *peer-to-peer* en tiempo real en aplicaciones de realidad mixta pensadas para ejecutarse sobre HoloLens 2 tal y como es el caso de este proyecto. Algunas características reseñables de MixedReality WebRTC son:

- Da la posibilidad de realizar comunicación bidireccional en tiempo real mediante audio, vídeo y datos, incluso existiendo múltiples pistas de cada uno de estos *streams*.
- Provee una interfaz abstracta para la señalización haciendo factible cambiar entre varias implementaciones. Expone una API para C++ y C# con el fin de poder ser integrada dentro de aplicaciones existentes.
- Contiene diversos componentes ya listos para su uso con el motor Unity, lo que facilita el prototipado rápido.
- Incluye soporte para ambas iteraciones del *headset* de Microsoft: HoloLens y HoloLens 2.
- Facilita el uso de Mixed Reality Capture (MRC) para poder hacer streaming del punto de vista del usuario de HoloLens 2.

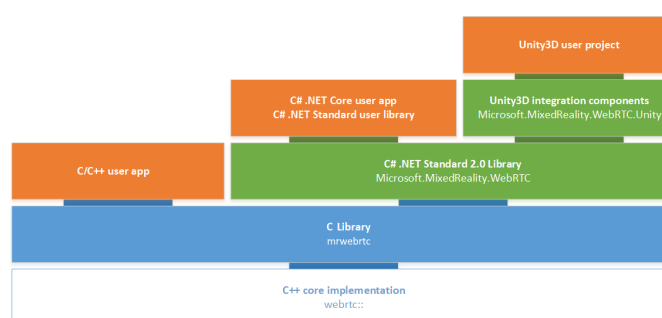


Figura 5.15: Arquitectura de MixedReality WebRTC[?]

⁴<https://github.com/microsoft/MixedReality-WebRTC>

El uso de estas utilidades se han preferido sobre el desarrollo de una solución propia debido a las razones siguientes:

- La gran cantidad de tiempo y recursos necesarios para una tarea de este tipo que podrían ser dedicados directamente a la implementación de las características propias y disruptivas del proyecto.
- Incluye todas las funcionalidades necesarias para el satisfactorio desarrollo del proyecto: transmisión bidireccional de audio, vídeo y datos.
- Su naturaleza de código abierto permitiría la modificación de sus propiedades y la implementación y mejora de nuevas funciones de ser esto necesario.
- Está desarrollada por la misma empresa que fabrica el *headset* HoloLens 2, lo que garantiza la compatibilidad.

5.3.2.2. Componentes más relevantes en la comunicación WebRTC

En esta sección se comentarán en detalle los aspectos más relevantes de la implementación de la parte WebRTC del proyecto. En el diagrama ?? se pueden ver de forma resumida los módulo y componentes de este subsistema.

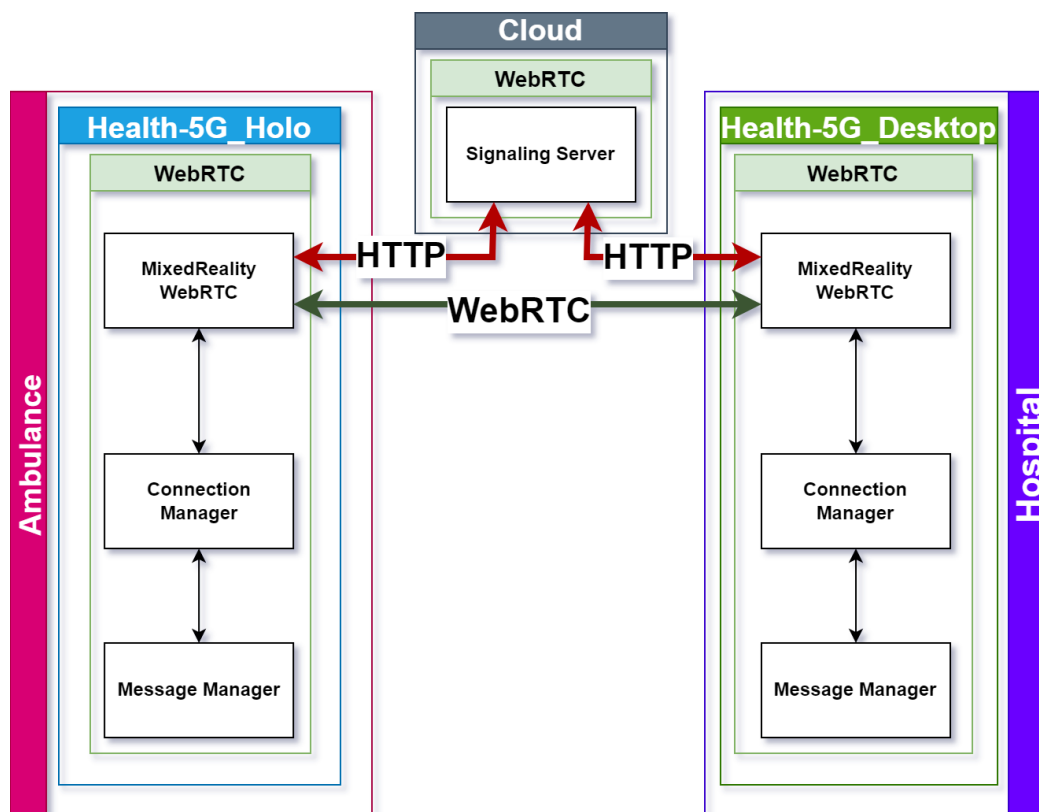


Figura 5.16: Sistema de comunicación bidireccional. Las flechas representan la dirección del intercambio de información.

Para el correcto funcionamiento de WebRTC es imprescindible un servidor de señalización (o *signaler*). En el caso de este proyecto se ha reutilizado por razones de agilidad en el desarrollo el servidor de señalización node-dss⁵.

⁵ <https://github.com/anpep/node-dss>

Por otro lado, la clase **DemoSignalingServer.cs** se ha realizado como una implementación en C# puro de un servidor de señalización WebRTC y funciona como un objeto de Unity más. No está pensada para usarse en producción y por eso no aparece en los diagramas, pero es extremadamente útil a la hora de hacer demostraciones fuera de línea del funcionamiento del producto evitando tener que ejecutar un servidor externo.

En cuanto a las aplicaciones Health-5G_Holo y Health-5G_Desktop, los módulos involucrados en la conexión WebRTC son MixedReality WebRTC, Connection Manager y Message Manager y son completamente iguales en forma y función entre ambos programas, compartiendo la totalidad del código fuente.

MixedReality WebRTC actúa a modo de “cajón de sastre” en el diagrama y representa de forma genérica a todos los componentes que incluye la biblioteca del mismo nombre para dar soporte al intercambio de audio, vídeo y datos a través de una conexión WebRTC. Los objetos más relevantes y usados en el proyecto son PeerConnection.cs y NodeDssSignaler.cs. La clase **PeerConnection.cs** representa una conexión WebRTC entre un dispositivo local y otro remoto y su responsabilidad es la de proveer diferentes métodos para iniciar las llamadas y manejar múltiples parámetros de las conexiones, tal y como puede ser establecer los códecs de vídeo y audio preferidos, identificar los transceptores que se van a usar, entre otros. Mixed Reality WebRTC ofrece la clase **NodeDssSignaler.cs** como un cliente de señalización para desarrollo y es lo que se ha usado en este proyecto, aunque hay que indicar que en un sistema de producción debería reemplazarse por otra implementación que soporte autenticación.

Sí bien la biblioteca utilizada incluye la mayoría de componentes necesarios para crear una conexión satisfactoria entre pares esto no significa que la parte de comunicación WebRTC haya sido directa de implementar en el proyecto. A lo largo del desarrollo se han observado múltiples problemas y pormenores comunes a los que hay que atender con el fin de crear una experiencia óptima para el usuario, entre los que podemos citar:

- Solo es posible añadir canales de datos de forma dinámica si antes de iniciar la llamada se ha creado al menos uno de estos canales. ⁶
- La única forma de colgar una llamada es desactivando y volviendo a activar el GameObject que contiene la conexión (PeerConnection). ⁷
- Un mensaje no podrá ser enviado antes de que se abran los canales de datos pues generará una excepción.
- No se proveen directamente eventos para indicar que una llamada ha iniciado correctamente, finalizado o fallado.

En este contexto surge la clase **ConnectionManager.cs** la cual encapsula, amplía y simplifica el funcionamiento de Mixed Reality WebRTC para permitir su utilización de forma sencilla en cualquier aplicación (de hecho se ha utilizado con éxito en otros proyectos comerciales). Esta clase expone métodos para iniciar o terminar la llamada, crear o retornar canales de datos y mandar mensajes, todos ellos implementados de forma que eliminan de forma completa los problemas enunciados anteriormente.

La clase **MessageManager.cs** extiende a ConnectionManager.cs con métodos específicos para la creación, apertura, cierre y mantenimiento de canales de mensajes WebRTC, así como para facilitar la redirección de dichos mensajes a los componentes que deberán recibirlos.

⁶<https://microsoft.github.io/MixedReality-WebRTC/api/Microsoft.MixedReality.WebRTC.PeerConnection.html>

⁷<https://github.com/microsoft/MixedReality-WebRTC/issues/375>

5.3.3. Interacción Persona-Ordenador

Después de extensivas pruebas se ha determinado que la forma más práctica de gestionar el emplazamiento de marcadores es mediante el trazado de un rayo, lo cual se explicará de forma técnica en posteriores apartados. El funcionamiento a grandes rasgos es el siguiente: cuando el especialista hace clic sobre cualquier parte de la pantalla se envía la posición del evento a las HoloLens y se coloca el marcador sobre la superficie más cercana en línea recta.

Esto permite que la posición del marcador en el espacio 3D sea completamente coherente con la posición del clic en el espacio 2D, lo que facilita la interacción y hace muy sencillo usar las diferentes herramientas. En la imagen ?? se pueden ver los componentes más importantes de este proceso.

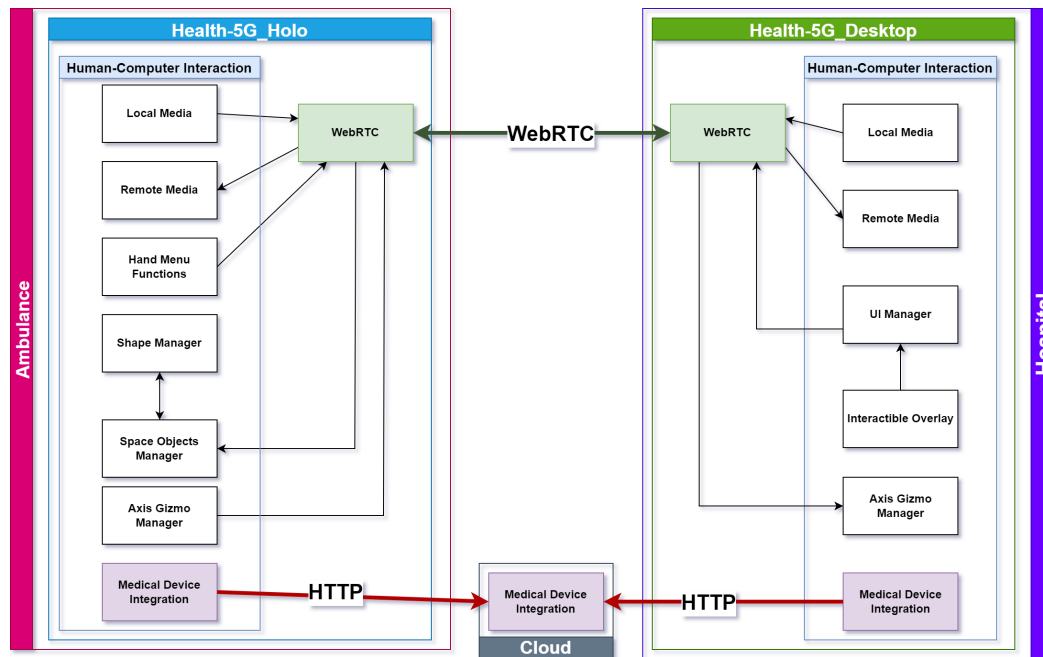


Figura 5.17: Sistema de interacción persona-ordenador. Las flechas representan la dirección del intercambio de información.

La interacción persona-ordenador incluye partes de los subsistemas vistos anteriormente, como la integración con dispositivos médicos y WebRTC, que se ha considerado relevante incluir en el diagrama pero que no se volverán a ver por haber sido explicados con anterioridad.

5.3.3.1. Medios locales y remotos

En el contexto de WebRTC, un transceptor (clase **Transceiver.cs**) es una especie de tubería metafórica que conecta los dispositivos locales y permite mandar audio y vídeo a través de ella. Cada transceptor tiene una dirección que indicará si está enviando o recibiendo algún medio, o si en caso contrario está inactivo. Por ejemplo, cuando se produce el envío de medios, la pista local del transceptor se utiliza como fuente y cuando se recibe algún medio éste se entrega a la pista remota del mismo.

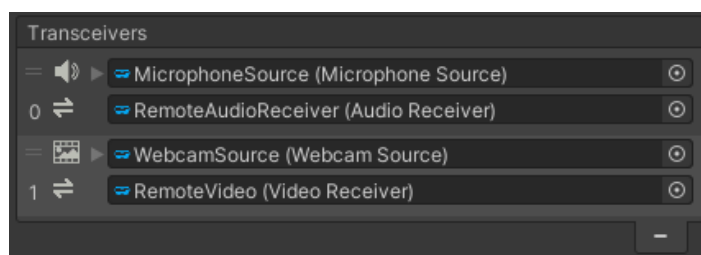


Figura 5.18: Ejemplo de transceptores usados en el proyecto

Para este proyecto se han usado los componentes (provistos por la biblioteca MixedReality WebRTC) `MicrophoneSource.cs`, `AudioReceiver.cs`, `WebcamSource.cs` y `VideoReceiver.cs` dentro de los transceptores de audio y vídeo, respectivamente. A continuación se ofrece una descripción breve de cada uno.

- **Local Media:** categoría que incluye a los componentes que se encargan de obtener audio y vídeo y enviar estos datos a través de WebRTC.
 - **MicrophoneSource.cs:** este componente representa una fuente de audio local que genera *frames* de audio desde un dispositivo de captura local (micrófono, en nuestro caso. Tanto el del PC como el de HoloLens 2).
 - **WebcamSource.cs:** este componente encapsula un emisor de vídeo local que produce fotogramas de vídeo desde un dispositivo de captura de vídeo local (cámaras web, en este caso).
- **Remote Media:** categoría que incluye a los componentes que se encargan de recibir audio y vídeo remotos a través de una conexión WebRTC y renderizarlos en la aplicación de destino.
 - **AudioReceiver.cs:** es un *endpoint* para una pista de audio remota que podrá ser después renderizada utilizando un componente de tipo **AudioRenderer.cs**.
 - **VideoReceiver.cs:** *endpoint* para una pista de vídeo remota será renderizada utilizando un componente del tipo **VideoRenderer.cs**.

De entre los anteriores componentes se debe hacer especial mención a `WebcamSource.cs` pues posibilita el uso de Mixed Reality Capture (MRC) en HoloLens 2. MRC es una tecnología que permite la superposición de los hologramas sobre la imagen real capturada por el *headset*. Esto hace que el médico especialista pueda recibir un flujo de vídeo que contiene de forma integral el punto de vista del equipo de emergencias.

5.3.3.2. Posicionamiento y manipulación de marcadores

El sistema más importante dentro de la interacción persona-ordenador es el que hace posible el emplazamiento de marcadores en el espacio 3D a partir de clics en una interfaz 2D. Los componentes que soportan este sistema son `InteractableOverlay.cs` y `UIManager.cs` en la aplicación de escritorio (Health-5G_Desktop) y `SpaceObjectsManager.cs` y `ShapeManager.cs` en la aplicación destinada a utilizarse en el *headset* HoloLens 2 (Health-5G_Holo).

InteractableOverlay.cs es el *script* que se encarga de registrar las acciones del usuario sobre la señal de vídeo que llega desde las gafas de realidad aumentada. En concreto es capaz de detectar tres eventos: cuando se hace clic izquierdo en el vídeo, cuando se arrastra el ratón sobre este y cuando termina el clic. Para poder efectuar esta detección se aprovechan los métodos de Unity `OnMouseDown()`, `OnMouseDownDrag()` y `OnMouseDownUp()` que son llamados en el momento en que se efectúa la acción apropiada. El método más importante de esta clase se encuentra en la función `CalculateNormalizedPoint()` la cual retorna un vector de dos dimensiones normalizado.

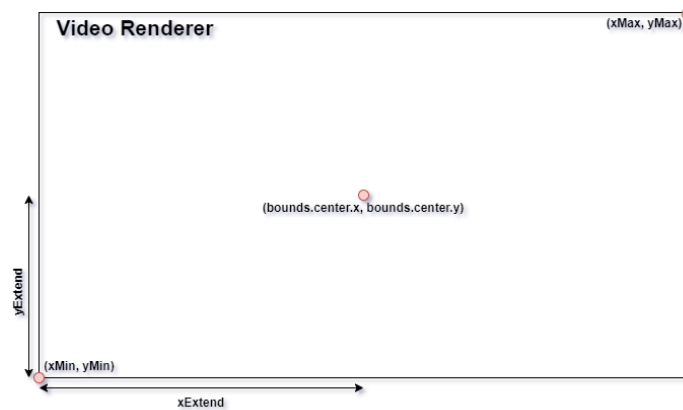
En el primer paso se lleva a cabo el cálculo de las coordenadas (en *world space*) correspondientes a los puntos mínimo y máximo del componente encargado de mostrar la señal de vídeo proveniente de HoloLens 2. Para esto contamos con el centro del objeto, representado por el punto (`bounds.center.x`, `bounds.center.y`) y la distancia de ese punto a cada lado del rectángulo (`xExtend` y `yExtend`).

Listado 5.5: Cálculo de puntos de interés para CalculateNormalizedPoint()

```

1 xExtend = videoRenderer.bounds.extents.x;
2 yExtend = videoRenderer.bounds.extents.y;
3 xMin = videoRenderer.bounds.center.x - xExtend;
4 yMin = videoRenderer.bounds.center.y - yExtend;
5 xMax = videoRenderer.bounds.center.x + xExtend;
6 yMax = videoRenderer.bounds.center.y + yExtend;

```

**Figura 5.19:** Puntos de interés para CalculateNormalizedPoint

El siguiente paso será calcular el punto (en *world space*) correspondiente al clic del usuario. Para esto, se obtiene primero a la distancia a la que está el vídeo de la cámara para después transformar las coordenadas del ratón en la pantalla y esta distancia en un punto del espacio 3D.

Listado 5.6: Cálculo del punto donde se ha hecho clic

```

1 screenPoint = mainCamera.WorldToScreenPoint(transform.position);
2 cursorScreenPoint = new Vector3(Input.mousePosition.x, ↵
  ↵ Input.mousePosition.y, screenPoint.z);
3 cursorPos = mainCamera.ScreenToWorldPoint(cursorScreenPoint);

```

En ultimo lugar se transforma el punto anteriormente calculado en unas coordenadas bidimensionales en intervalo $[0, 1]$ y se retorna.

Listado 5.7: Normalización de las coordenada

```

1 return new Vector2((cursorPos.x - xMin) / (xMax - xMin), ↵
  ↵ (cursorPos.y - yMin) / (yMax - yMin));

```

La clase **UIManager.cs** de la aplicación para computador de escritorio actúa a modo de intermediario entre la interfaz de usuario y WebRTC. Entre sus responsabilidades se encuentran generar los mensajes correspondientes a una pulsación sobre el vídeo o un botón y enviarlos a través de MessageManager.cs o de iniciar o terminar la llamada con la ayuda de ConnectionManager.cs cuando se pulsa el botón correspondiente.

Los tipos de mensajes utilizados en la interacción son dos: **ShapeModifiersMessage.cs** y **ClickedVideoMessage.cs**. El primero de estos se enviará cuando se pulse cualquiera de los botones de herramienta de la interfaz (escala, posición, seleccionar siguiente objeto, seleccionar anterior objeto, restaurar posición y escala y borrar marcador), mientras que el mensaje ClickedVideoMessage.cs se envía cuando se interactúa con la señal de vídeo proveniente del dispositivo HoloLens 2. El formato del mensaje es el siguiente:

Listado 5.8: Estructura del mensaje ClickedVideoMessage

```

1 public class ClickedVideoMessage
2 {
3     // ID del vídeo sobre el que se ha hecho clic
4     public string videoId;
5
6     // ID de la forma que se quiere poner
7     // Arrow = 0, Box = 1, Cross = 2, Line = 3
8     public int shapeId;
9
10    // Coordenada 'x' en el intervalo [0, 1]
11    public float x;
12
13    // Coordenada 'y' en el intervalo [0, 1]
14    public float y;
15
16    // Distancia por defecto a la que colocar el marcador
17    // si el raycasting no surte efecto
18    public float noHitObjectDistance;
19
20    // Acción del mouse
21    // Down = 0, Drag = 1, Up = 2
22    public int mouseActionId;
23 }

```

Estos mensajes llegarán a la aplicación de HoloLens 2 a través de WebRTC y serán interpretados por la clase **SpaceObjectsManager.cs**, cuya responsabilidad es la de colocar los marcadores en el espacio y efectuar sobre ellos las modificaciones que elija el usuario de escritorio. El ciclo de vida de un marcador comienza con su instanciación por parte de la clase **ShapeManager.cs** a petición de SpaceObjectsManager.cs.

El siguiente paso será tomar la posición normalizada que representa el clic del usuario y transformarla de forma que coincida con la resolución de renderizado de HoloLens 2:

Listado 5.9: Transformación a coordenadas de pantalla de HoloLens 2

```

1 screenSpaceVector.x = clickedVideoMessage.x * matrixWidth + ↩
   ↩ screenSpaceVector.OffsetX;
2 screenSpaceVector.y = clickedVideoMessage.y * matrixHeight + ↩
   ↩ screenSpaceVector.OffsetY;

```

La variable *matrixWidth* tiene un valor de 1440 y es la resolución de renderizado horizontal de HoloLens 2, mientras que *matrixHeight* es la altura en píxeles de una imagen con formato 16:9 y la anchura mencionada al principio. Si bien la resolución de renderizado del *headset* no tiene una relación de aspecto 16:9 sí que la tiene la señal de vídeo capturada por la cámara RGB. Las variables *screenSpaceVector.OffsetX* y *screenSpaceVector.OffsetY* son la diferencia de posición entre la cámara virtual que renderiza los hologramas y la cámara RGB y tienen un valor de 0 y -25 respectivamente.

Este vector se podrá transformar en un rayo con origen en la cámara principal de Unity usando el método incorporado *ScreenPointToRay(Vector3 pos, Camera.MonoOrStereoscopicEye eye)*:

Listado 5.10: Transformación de coordenadas de pantalla a rayo

```

1 mainCamera.ScreenPointToRay(screenSpaceVector, ↩
   ↩ Camera.MonoOrStereoscopicEye.Mono);

```

Si el marcador generado corresponde a una flecha, una caja o una cruz se realizarán un total de nueve *raycasts* desde la cámara a la malla que representa el espacio físico en el que se encuentra el usuario con una muy ligera dispersión formando una cuadrícula tal y como se muestra en la imagen ??.

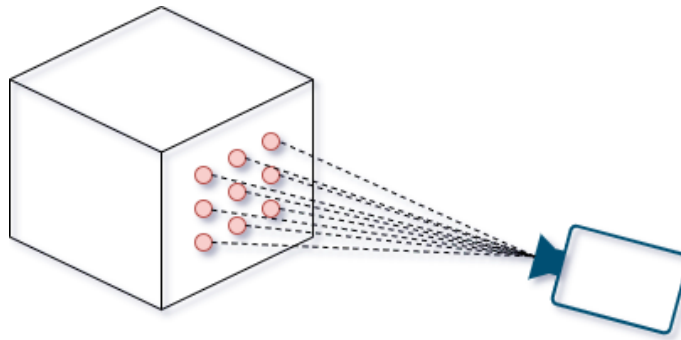


Figura 5.20: Raycasts hacia el espacio físico

El marcador se colocará a lo largo del rayo central (que tiene dispersión cero) a una distancia que corresponde a una media aritmética de las distancias entre la cámara y el punto de colisión de cada uno de los rayos generados que hayan impactado con la malla que representa el espacio físico y con una rotación correspondiente a la media de las normales de las superficies impactadas.

En caso de que ningún rayo haya impactado con el entorno físico, se procederá a emplazar el marcador a la media de la distancia de los marcadores que se encuentren en la misma dirección. Esto permite aproximar un posicionamiento acertado de los marcadores aún cuando haya agujeros u otras irregularidades en la malla que representa el espacio físico. Si no se encontrara ningún marcador en la misma dirección que el marcador nuevo se procederá a colocarlo a una distancia predeterminada.

Si el marcador corresponde a un dibujo libre el procedimiento será similar solo que por cada punto que llegue se ejecutará un solo trazado de rayos, por motivos de rendimiento, y se dibujarán líneas entre los puntos para simular un trazo continuo. Una vez acabado el dibujo se generará una malla de colisión que lo envuelva completamente para posibilitar la manipulación del mismo mediante gestos por parte del usuario del *headset*.

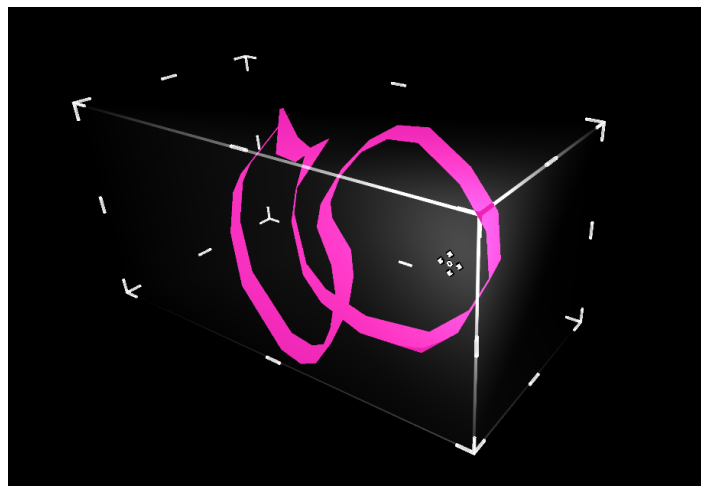


Figura 5.21: Dibujo libre envuelto por una malla de colisión

Sea cual sea la forma del marcador, tras crearlo se procede a guardar su posición, escala y rotación inicial en un objeto de tipo **InitialPositionAndScale.cs**. Esto permitirá restablecer las transformaciones del marcador a sus valores por defecto cuando se use la herramienta apropiada (*reset*).

Una vez que el marcador deseado por el usuario está en la escena se podrán hacer diferentes acciones sobre él, las cuales llegarán en la forma de mensajes **ShapeModifiersMessage.cs** con la siguiente estructura:

Listado 5.11: Estructura del mensaje ShapeModifiersMessage

```

1 public class ShapeModifiersMessage
2 {
3     // Herramienta seleccionada
4     // Scale = 0, Position = 2, SelectNext = 3,
5     // SelectPrevious = 4, Reset = 5, Delete = 6
6     public int selectedTool;
7
8     // Las siguientes variables solo son útiles si la herramienta
9     // seleccionada es Scale o Position
10
11     // Modificador
12     // More = 0, Less = 1
13     public int selectedModifier;
14
15     // Aplicar transformaciones en el eje X
16     public bool selectedXAxis;
17
18     // Aplicar transformaciones en el eje Y
19     public bool selectedYAxis;
20
21     // Aplicar transformaciones en el eje Z
22     public bool selectedZAxis;
23 }

```

La herramienta de escala funciona escalando positiva o negativamente el objeto seleccionado en los ejes determinados por el usuario, mientras que la herramienta de posicionado es similar pero cambiando la ubicación del objeto en el espacio. El código de este procedimiento se incluye en el listado ??.

Listado 5.12: Modificación de escala o posición de marcadores

```

1 float step = (Modifier)ShapeModifiersMessage.selectedModifier
2     == Modifier.More ? Mathf.Abs(scaleStep) :
3     Mathf.Abs(scaleStep) * -1;
4 float xValue = ShapeModifiersMessage.selectedXAxis ? step : 0;
5 float yValue = ShapeModifiersMessage.selectedYAxis ? step : 0;
6 float zValue = ShapeModifiersMessage.selectedZAxis ? step : 0;
7 if (selectedTool == Tool.Scale)
8 {
9     selectedObject.transform.localScale = new
10         Vector3(Mathf.Clamp(selectedObject.transform.localScale.x
11             + xValue, minScale, maxScale),
12             Mathf.Clamp(selectedObject.transform.localScale.y +
13                 yValue, minScale, maxScale),
14             Mathf.Clamp(selectedObject.transform.localScale.z +
15                 zValue, minScale, maxScale));
16 }
17 else if (selectedTool == Tool.Position)
18 {
19     selectedObject.transform.position = new
20         Vector3(selectedObject.transform.position.x + xValue,
21             selectedObject.transform.position.y + yValue,
22             selectedObject.transform.position.z + zValue);
23 }

```

La herramienta *reset* hace uso del *script* `InitialPositionAndScale.cs` mencionado anteriormente mientras que las utilidades de selección de marcador siguiente, anterior y borrado utilizan la clase `ShapeManager.cs` pues guarda una lista de todos los marcadores, así como el que está seleccionado en un instante determinado. Las herramientas *siguiente* y *anterior* simplemente cambian el marcador seleccionado y modifican el material de este para que tenga un color que indica al usuario que ese es el objeto con el que se va a interactuar, mientras que la utilidad de borrado destruye el marcador y selecciona el siguiente en la lista.

5.3.3.3. Otros componentes de la interfaz

Algunos componentes secundarios de la interfaz de usuario incluyen el menú mano, el indicador de orientación y los comandos de voz.

El *menú mano* es una colección de accesos rápidos que se encuentra en la aplicación de HoloLens 2 y dispone de botones a funciones convenientes que el usuario puede activar si pone la palma de su mano en dirección a su cara. Estos botones tienen las funciones de mostrar y ocultar la interfaz, centrar los hologramas, indicar a los hologramas que sigan o dejen de seguir al usuario y finalmente, de iniciar o acabar una llamada.



Figura 5.22: El menú mano

Este menú hace uso de la clase **HandMenuFunctions.cs** para llevar a cabo sus funciones, procedimientos que se exponen públicamente y pueden ser llamados desde un objeto de tipo **SpeechInputHandler.cs** (provisto por MRTK) para dar soporte a los comandos de voz.

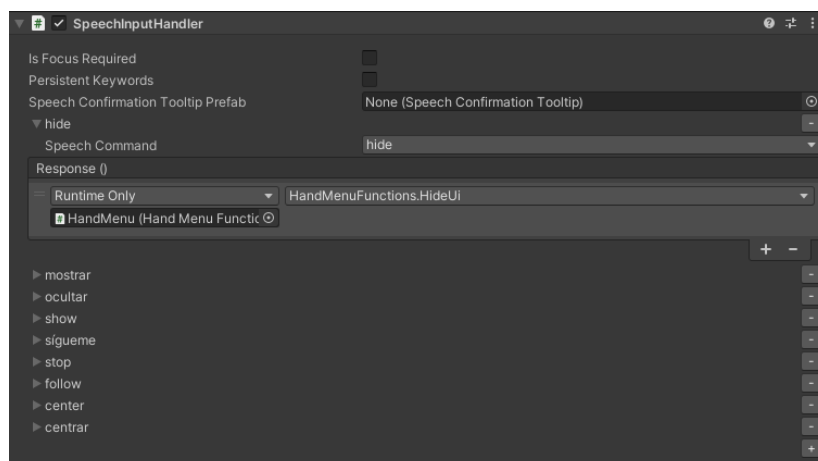


Figura 5.23: SpeechInputHandler usado por el menú mano

Una parte importante de la interfaz de usuario de la aplicación de escritorio es el indicador de orientación que muestra la dirección a la que mira la persona que lleva puesta el *headset*.



Figura 5.24: Indicador de orientación en Health-5G_Desktop

Este componente hace uso de la clase **AxisGizmoManager.cs** tanto en la aplicación de escritorio como en la de HoloLens 2, siendo la única diferencia que en una está configurada para enviar la orientación de la cámara principal de Unity y en la otra para recibirla y girar el objeto al que pertenece el *script* para modificar su orientación y hacer que esta coincida con los datos recibidos.

5.3.4. Otras partes de la arquitectura

En esta última sección se incluyen algunos componentes más de la arquitectura, que si bien no tienen suficiente relevancia para tener un apartado propio sí que son de interés para el proyecto:

LoggerManager.cs: Unity no desactiva por defecto la impresión de mensajes de *debug* en las versiones finales de la aplicación, aunque el usuario final no pueda verlos al no haber consola donde mostrarlos. En PC el impacto en cuanto a rendimiento de esta situación es insignificante, más no así en HoloLens 2 cuya capacidad de procesamiento limitado hace necesarios todos los esfuerzos de optimización posibles.

Listado 5.13: Código de Logger Manager

```

1 public class LoggerManager : MonoBehaviour
2 {
3     private void Awake()
4     {
5         #if UNITY_EDITOR
6             Debug.unityLogger.logEnabled = true;
7         #else
8             Debug.unityLogger.logEnabled = false;
9         #endif
10    }
11 }
```

El código del listado ?? aprovecha la compilación condicional para desactivar el *logger* de Unity fuera del editor, lo que ha demostrado aumentar de forma muy apreciable el rendimiento de la aplicación en el *headset*.

Por otra parte, la mayoría de funciones de Unity sólo pueden ejecutarse en el hilo principal de la aplicación y se producirán excepciones y cuelgues si se llama a cualquiera de estas funciones desde otro hilo. **MainThreadManager.cs** ha sido la solución a esta problemática pues garantiza que un código será ejecutado en el hilo principal.

Listado 5.14: Código de Main Thread Manager

```

1 internal class MainThreadManager : MonoBehaviour
2 {
3     internal static MainThreadManager manager;
4     private Queue<Action> jobs = new Queue<Action>();
5
6     private void Awake()
```

```

7      {
8          manager = this;
9      }
10     private void Update()
11     {
12         while (jobs.Count > 0)
13         {
14             jobs.Dequeue().Invoke();
15         }
16     }
17     internal void AddJob(Action newJob)
18     {
19         jobs.Enqueue(newJob);
20     }
21 }

```

La función *Update* es propia del *MonoBehaviour* de Unity y es llamada en cada *frame*, siempre desde el hilo principal, por lo que si hay trabajos pendientes en la cola solamente habrá que desencolarlos y ejecutarlos. Esto permite casos de uso tan interesantes como el siguiente:

Listado 5.15: Uso de Main Thread Manager

```

1 private void IceStateChangedDelegate(IceConnectionState newState)
2 {
3     switch (newState)
4     {
5         ...
6         case IceConnectionState.Connected:
7             if (!onCall)
8             {
9                 onCall = true;
10                // Lanzamos el evento de inicio de llamada
11                MainThreadManager.manager.AddJob(() =>
12                {
13                    callStarted.Invoke();
14                });
15            }
16            break;
17        ...
18    }
19 }

```

El delegado *IceStateChangedDelegate* será ejecutado desde un hilo cualquiera, lo que podría provocar un cuelgue de la aplicación si alguno de los subscriptores al evento *callStarted* usa una función de Unity como *SetActive(bool value)* (activa o desactiva un *GameObject*). Al añadir la invocación a la cola de trabajo de Main Thread Manager se asegura que este código se ejecute desde el hilo principal sin problemas.

También podemos citar como un elemento interesante de la arquitectura a las clases *ConfigManager.cs* y *ConfigurableMonoBehaviour.cs*, las cuales usan la biblioteca *SharpConfig*⁸ con el fin de facilitar el uso de archivos de configuración basados en texto para permitir el ajuste de parámetros importantes de las aplicaciones de videoconferencia.

⁸<https://github.com/cemdervis/SharpConfig>

Listado 5.16: Archivo de configuración de la aplicación de escritorio

```

1  [InteractiveOverlay]
2  minimumDragDistance=0.001
3
4  [PeerConnection]
5  stunServer=
6  preferredVideoCodec=H264
7  minBitrateMbps=5
8  startBitrateMbps=5
9  maxBitrateMbps=15
10
11 [UIManager]
12 videoId=default
13 minDistance=1.4
14
15 [ImageDownloadManagerHub]
16 url=http://img.health-5g-server.api.5gclm.es/
17 startVideoOnStartup=False
18
19 [NodeDssSignaler]
20 httpServerAddress=http://127.0.0.1:3000
21 localPeerId=COMPUTER
22 remotePeerId=HOLOLENS

```

ConfigManager.cs actúa a modo de *manager* cuya función es la de buscar en la escena todos los *scripts* que hereden de la clase **ConfigurableMonoBehaviour.cs** y cargar la configuración si existe un archivo `config.cfg` apropiado o crearlo con los valores por defecto si este no es encontrado. En el listado ?? se encuentra un ejemplo comentado de una clase que hereda de **ConfigurableMonoBehaviour.cs**.

Listado 5.17: Ejemplo de clase que implementa **ConfigurableMonoBehaviour**

```

1  public class InteractiveOverlay : ConfigurableMonoBehaviour
2  {
3      // Llamado por ConfigManager si no encuentra un archivo config.cfg.
4      // Devuelve un objeto de tipo Configuration con los valores por defecto
5      // de esta clase.
6      public override void SetupCleanConfiguration(Configuration cfg)
7      {
8          cfg["InteractiveOverlay"]["minimumDragDistance"]
9              .FloatValue = minimumDragDistance;
10     }
11
12     // Una vez que el archivo config.cfg se encuentra disponible,
13     // ConfigManager se asegurará de llamar a LoadConfiguration con
14     // el objeto Configuration procedente de parsear el archivo
15     // config.cfg
16     public override void LoadConfiguration(Configuration cfg)
17     {
18         ParseConfigFloat(
19             cfg["InteractiveOverlay"]["minimumDragDistance"],
20             ref minimumDragDistance);
21     }
22
23     ...
24 }

```

5.4. MEDICIONES DE LATENCIA

Una de las medidas de rendimiento más útiles que ofrece WebRTC es el *round-trip time* (RTT), que mide el *jitter* o la latencia de un *frame* a otro. Estos detalles están expuestos en el API de bajo nivel de WebRTC por lo que no son accesibles desde MixedReality WebRTC. Esta situación ha provocado que se deban buscar maneras alternativas de medir el tiempo que pasa desde que se envía un cuadro de vídeo hasta que llega a su destino.

Al final se ha determinado que una forma sencilla y realista de llevar a cabo estos experimentos consiste en enviar un vídeo desde la aplicación del especialista en PC a la del equipo de emergencias en HoloLens 2, siendo esta última convenientemente modificada para que reenvíe al origen los cuadros que recibe.

Las imágenes del vídeo de prueba contienen una serie de bosques y paisajes en movimiento con un código de tiempo en la esquina inferior derecha. Restando el código de tiempo del *frame* que se está enviando actualmente del que se recibe desde HoloLens 2 y dividiendo entre dos el resultado podemos hacernos una idea en segundos de la latencia existente.

Cómo el códec de vídeo es modificable por el usuario del sistema final, se han realizado experimentos con los códecs de vídeo H.264, VP8 y VP9, utilizando el mismo vídeo a resolución 640x480px y tomando mediciones de latencia a intervalos regulares a lo largo de una videoconferencia de 70 segundos. En todos los casos el códec de audio es opus con una frecuencia de muestreo de 48kHz y dos canales de sonido. Se pueden observar a continuación los resultados:

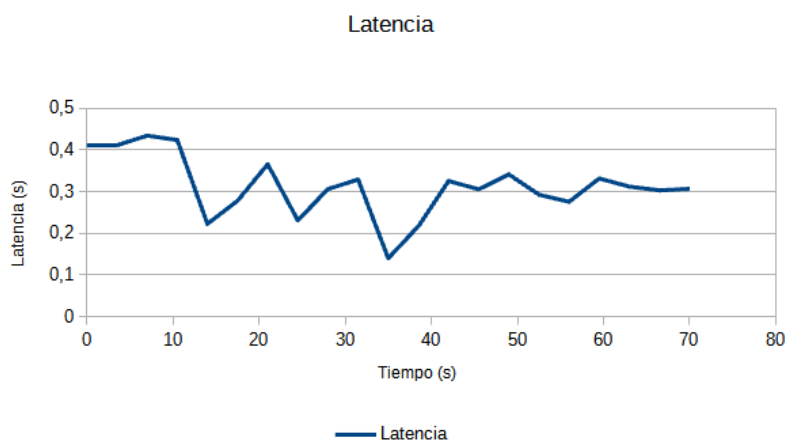


Figura 5.25: Latencia con el códec H.264

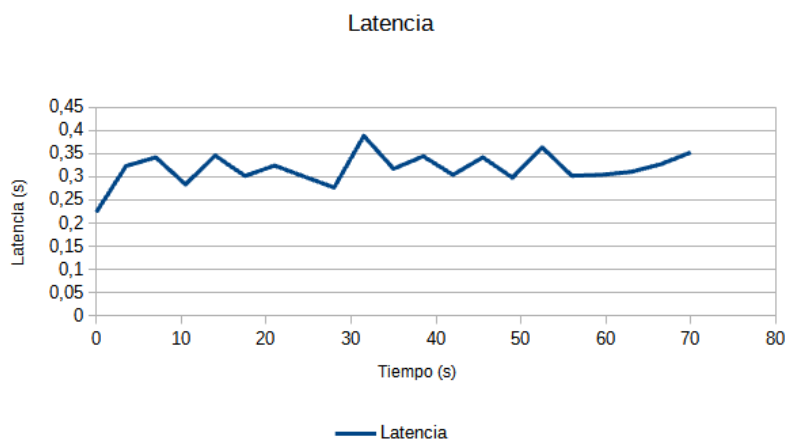


Figura 5.26: Latencia con el códec VP8

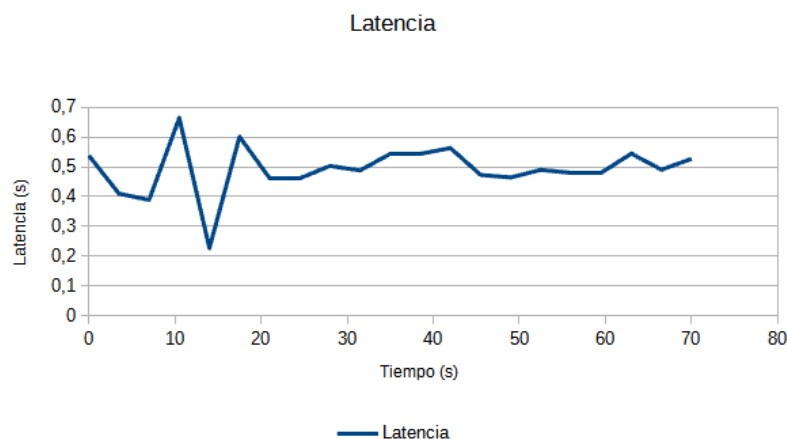


Figura 5.27: Latencia con el códec VP9

Las latencias medias obtenidas han sido de 0,3125s para el códec H.264, 0,3176s para VP8 y 0,4922s para VP9. Se pueden ver representadas gráficamente en la figura ??.

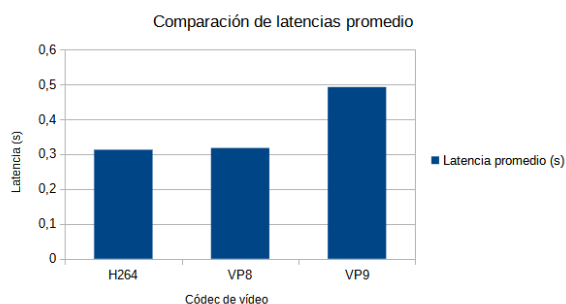


Figura 5.28: Promedio de las latencias obtenidas

Se han obtenido muy buenos tiempos en los tres casos consiguiendo siempre latencias de menos de medio segundo. Se ha decido usar el códec H.264 por defecto por su ventaja en estas pruebas y por ser soportado por el decodificador hardware de HoloLens 2, haciendo que su procesamiento sea más eficiente que el de VP8 y VP9 el cual debe llevarse a cabo mediante software.

5.5. ESTADÍSTICAS DE RENDIMIENTO

Es un requisito indispensable mantener un rendimiento adecuado en las aplicaciones de videoconferencia para convertir la experiencia de comunicación en algo útil para los usuarios, sobretudo en el caso de la parte que funciona en el *headset* HoloLens 2 pues una tasa baja de fotogramas por segundo (FPS) podría causar mareos, dolores de cabeza, náuseas y otros problemas completamente inaceptables en el entorno de la medicina de emergencias.

A la hora de medir el rendimiento se han efectuado tres conferencias de 70 segundos con un diferente número de marcadores en la escena hasta un máximo de 50, muy por encima de los que se llegarán a usar en un entorno real. Para las siguientes pruebas definiremos una tasa aceptable de fotogramas por segundo en torno a 30FPS o más en PC y alrededor de 60FPS o más en HoloLens 2.

5.5.1. Rendimiento con 15 marcadores holográficos

Se han empezado los experimentos con 15 marcadores holográficos y se ha medido el rendimiento de la aplicación en sí (línea azul) y el rendimiento de la videoconferencia (línea naranja). Se comienza con los datos obtenidos en HoloLens 2:

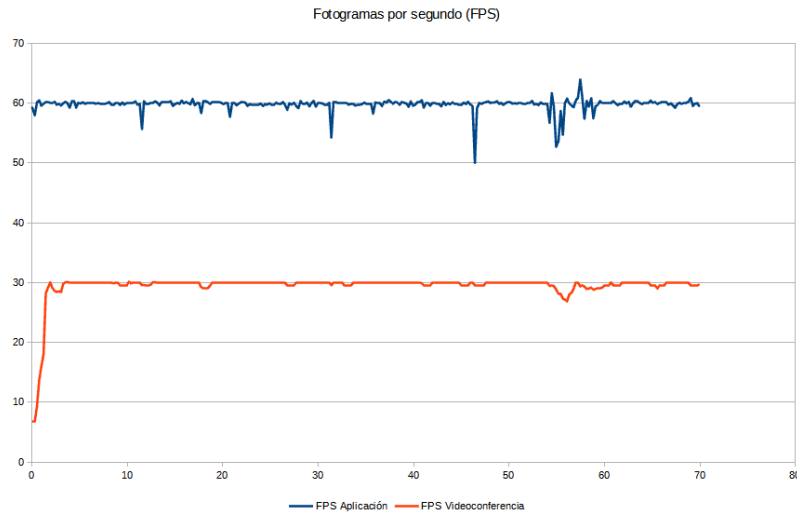


Figura 5.29: Fotogramas por segundo con 15 marcadores (HoloLens 2)

La media de FPS de la aplicación y del vídeo proveniente del PC con esta configuración ha sido de 59,81FPS y 29,41FPS respectivamente, cumpliendo los objetivos deseados. En PC los resultados obtenidos han sido los siguientes:

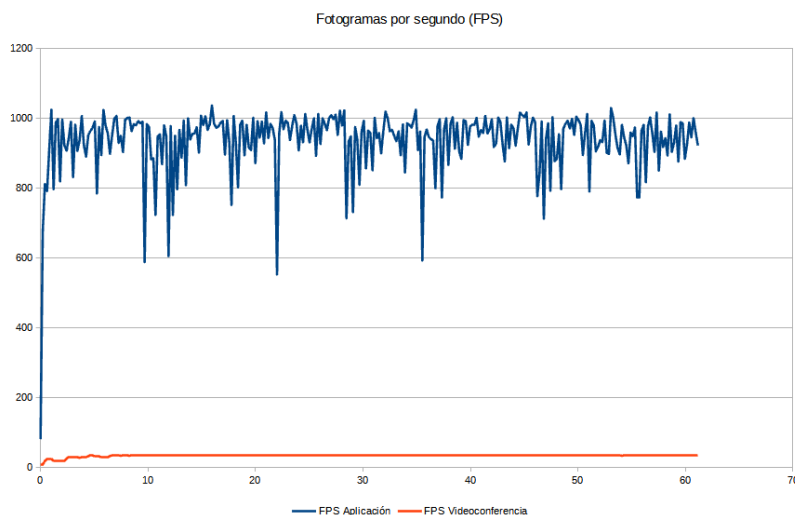


Figura 5.30: Fotogramas por segundo con 15 marcadores (PC)

Por un lado la aplicación en sí obtiene una tasa de refresco media de 936,23FPS y el vídeo que llega desde HoloLens 2 se renderiza a una media de 31,56FPS.

5.5.2. Rendimiento con 30 marcadores holográficos

Para la siguiente experiencia se han duplicado los marcadores, para un total de 30. Se comienza mostrando los datos obtenidos en HoloLens 2:

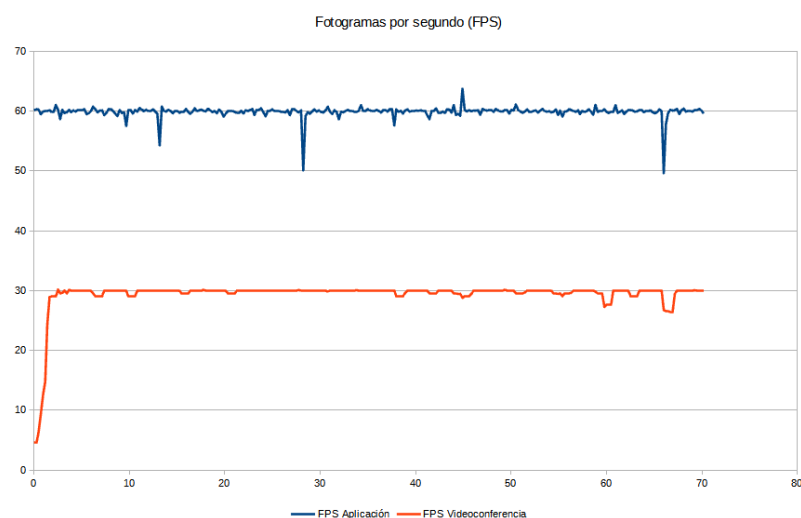


Figura 5.31: Fotogramas por segundo con 30 marcadores (HoloLens 2)

La media de FPS de la aplicación y del vídeo proveniente del PC con esta configuración ha sido de 59,84FPS y 29,32FPS, cumpliendo por segunda vez los objetivos deseados. En PC los resultados obtenidos han sido los siguientes:

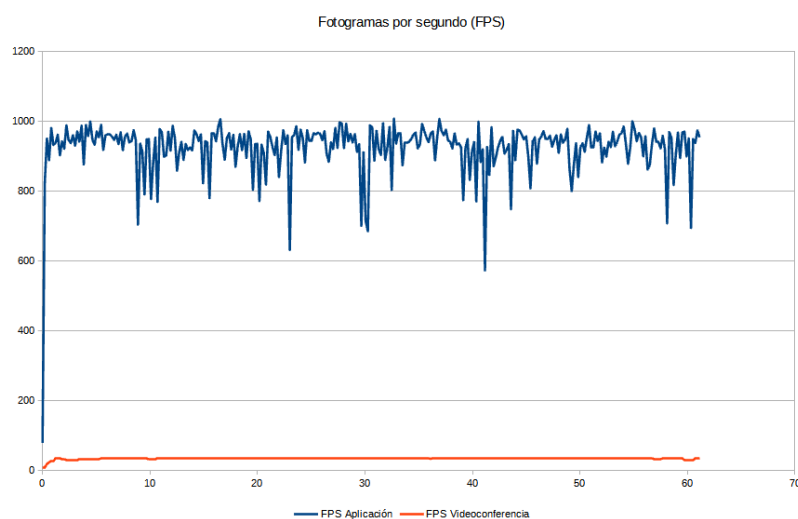


Figura 5.32: Fotogramas por segundo con 30 marcadores (PC)

Por un lado la aplicación en sí obtiene una tasa de refresco media de 923,6FPS y el vídeo que llega desde HoloLens 2 se renderiza a una media de 31,93FPS.

5.5.3. Rendimiento con 50 marcadores holográficos

Para la prueba final se ha aumentado el número de marcadores a 50, más de los que se llegarían a usar en un entorno de producción real típico. Se comienza con los datos obtenidos en HoloLens 2:

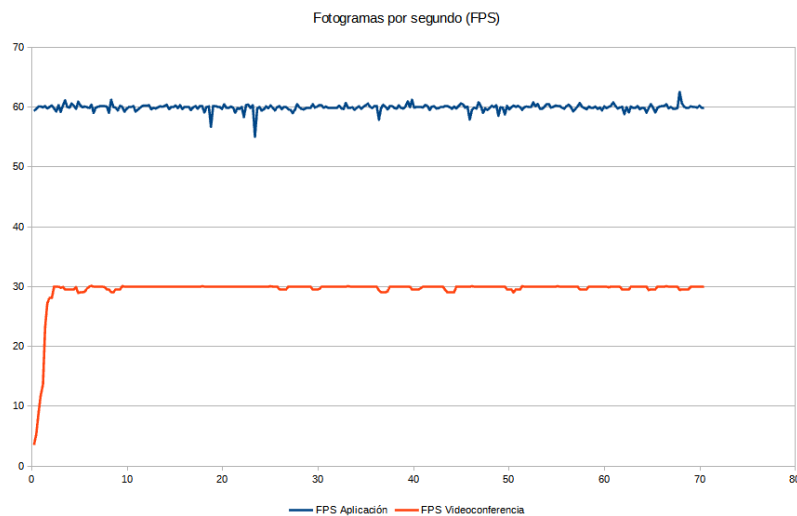


Figura 5.33: Fotogramas por segundo con 50 marcadores (HoloLens 2)

La media de FPS de la aplicación y del vídeo proveniente del PC con esta configuración ha sido de 59,95FPS y 29,47FPS, cumpliendo otra vez los objetivos. En PC los resultados obtenidos han sido:

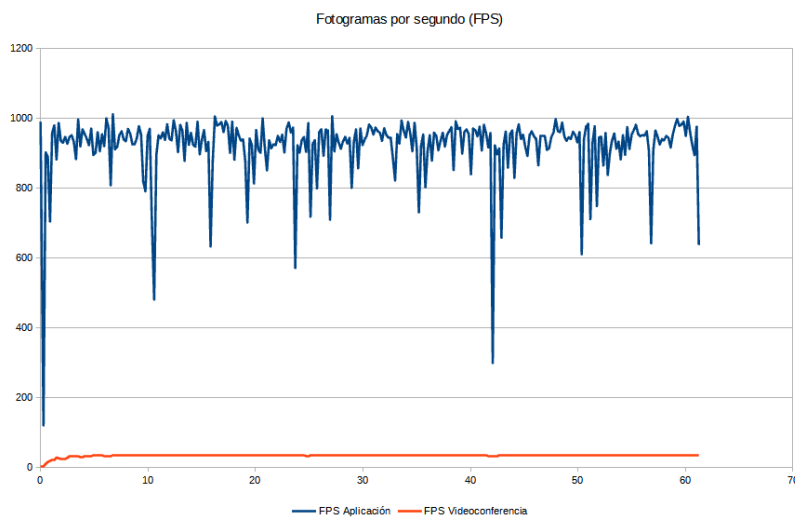


Figura 5.34: Fotogramas por segundo con 50 marcadores (PC)

Por un lado la aplicación en sí obtiene una tasa de refresco media de 920,4FPS y el vídeo que llega desde HoloLens 2 se renderiza a una media de 31,69FPS.

5.6. COSTE ECONÓMICO DEL PROYECTO

Este proyecto se ha desarrollado entre los días 5 de agosto de 2021 y 23 de noviembre de 2022, lo cual puede parecer un periodo de tiempo considerable pero hay que tener en cuenta que a la vez se han tenido que realizar otras actividades laborales y académicas, por lo que la dedicación no ha sido a jornada completa. En general se estima una dedicación de unas 400 horas a la implementación de los proyectos software unidas a unas 150 de escritura de la memoria. Suponiendo un sueldo de desarrollador de 38,17€⁹ la hora (y sin tener en cuenta el tiempo que ha llevado realizar esta memoria) podemos estimar los siguientes costes:

Descripción	Cantidad	Coste
Sueldo desarrollador	1	15.268€
Microsoft HoloLens 2	1	3.849€
MSI MS-16RU	1	1.150€
TP-Link Archer AX10	1	59,99€
Total		20.326,99€

Tabla 5.9: Desglose del coste del proyecto

Se han realizado 113 *commits* al repositorio durante el desarrollo del proyecto, lo que tras múltiples esfuerzos de refactorización y optimización resulta finalmente en una extensión de 5214 líneas de código no contando líneas en blanco. De estas líneas son comentarios un 32 %, y están distribuidas a lo largo de 43 archivos (las clases compartidas entre proyectos solo se están contando una vez). Para contar las líneas se ha usado la herramienta *cloc*¹⁰ en las carpetas relevantes. En la tabla ?? aparecen los números concretos.

Language	Files	Blank	Comment	Code
C#	34	700	1608	3023
Python	3	52	40	302
HLSL	4	39	3	218
Dockerfile	1	8	0	11
YAML	1	1	0	9
SUM	43	800	1651	3563

Tabla 5.10: Estadísticas de número de líneas

Basándonos en los datos de las tablas anteriores se puede estimar un precio por línea de código de 2,93€. Este resultado se obtiene tras dividir el coste del desarrollador entre el número de líneas totales sin incluir aquellas en blanco.

⁹https://www.payscale.com/research/US/Job=Unity_Developer/Salary

¹⁰<https://github.com/AlDanial/cloc>

Conclusiones

En este capítulo se van a justificar las competencias adquiridas durante el desarrollo de este trabajo para a continuación repasar si se ha cumplido cada uno de los objetivos del mismo. Finalmente se darán algunas líneas de trabajo futuro.

6.1. JUSTIFICACIÓN DE LAS COMPETENCIAS ADQUIRIDAS

Las competencias adquiridas a lo largo de estos meses han sido complementarias a las que se desarrollaron en el trabajo de fin de grado¹, que utilizó una tecnología similar pero enfocada en la enseñanza de la programación a escolares. Sin embargo, el enfoque más orientado a la investigación de este proyecto y el trabajo a más bajo nivel ha requerido mayor destreza en el diseño de la arquitectura y ha supuesto un nivel de dificultad significativamente mayor.

En la tabla ?? se exponen las competencias adquiridas así como una justificación de las mismas.

Competencia	Justificación
[CE4] Capacidad para modelar, diseñar, definir la arquitectura, implantar, gestionar, operar, administrar y mantener aplicaciones, redes, sistemas, servicios y contenidos informáticos.	El núcleo del proyecto ha consistido en el diseño y desarrollo de un sistema distribuido formado por múltiples aplicaciones para plataformas diversas, así como su despliegue y mantenimiento.
[CE14] Capacidad para conceptualizar, diseñar, desarrollar y evaluar la interacción persona-ordenador de productos, sistemas, aplicaciones y servicios informáticos	Una parte muy importante del desarrollo de este trabajo ha tratado sobre el diseño de unas interfaces persona-ordenador lo suficientemente intuitivas y ágiles para permitir su uso en un momento de emergencia. Además, las interfaces desarrolladas han sido heterogéneas y destinadas a diferentes plataformas con distintos paradigmas de interacción, principalmente computador de escritorio y realidad aumentada.

Tabla 6.1: Justificación de las competencias específicas abordadas en el TFM

¹<https://github.com/OhEsPaco/RoboTIC>

6.2. CONCLUSIONES EN CUANTO AL RENDIMIENTO

Una parte muy importante de cualquier sistema cuyo eje central es la comunicación en tiempo real es la de mantener una baja latencia, lo que se ha alcanzado gracias al uso de WebRTC. Tal y como se expone en el apartado ?? sobre mediciones de latencia, se han obtenido unos resultados muy satisfactorios con 300ms en promedio.

Por otro lado, en la sección ?? sobre estadísticas de rendimiento se aprecia que la aplicación para PC cumple (e incluso supera) los objetivos de rendimiento, aunque por supuesto estos números dependen exclusivamente de las especificaciones hardware del mismo, lo cual es un factor variable e impredecible, ya que dependerá del equipo del especialista donde se ejecute el cliente. En el lado del personal técnico de la GUETS no será dependiente, pues utilizará el dispositivo con el que se han realizado las pruebas (HoloLens 2).

La aplicación para el *headset* (Health-5G_Holo) se ha mantenido en todos los experimentos rondando la tasa de 60FPS con 30FPS en el renderizado de vídeo procedente de la webcam del especialista, lo cual está dentro de los objetivos deseados. Además, tal y como se mencionó anteriormente, HoloLens 2 incluye un hardware cerrado y estándar a lo largo de todos los dispositivos del mismo modelo por lo que se puede garantizar que el usuario inmerso en la realidad aumentada no tendrá problemas de mareos, dolores de cabeza ni desorientación como resultado del uso del programa.

6.3. CONCLUSIONES DE DISEÑO Y ARQUITECTURA

A la hora de embarcarnos en un proyecto de esta envergadura es necesario primero plantearnos dos preguntas: *qué queremos hacer* y *cómo lo vamos a hacer*. Ambas quedan reflejadas en los objetivos parciales de este trabajo de fin de máster.

La primera de estas cuestiones «*qué*» implica tomar las ideas abstractas que han dado pie al proyecto y enfrentarlas a las conclusiones de otros trabajos de temática similar, con el fin de saber qué es lo que ya se ha hecho y así poder aportar algo que mejore el estado del arte actual. Además nos permitirá identificar errores que han cometido otros investigadores en trabajos previos.

Para cumplir con esto se ha realizado un pequeño análisis de los estudios relacionados más importantes. Su influencia en este trabajo es evidente: los marcadores como base para la interacción, el uso comandos de voz y la videoconferencia son temas recurrentes en este campo.

La segunda cuestión «*cómo*» es el paso previo a convertir estas ideas abstractas en realidad teniendo claras las tecnologías y métodos de desarrollo que soportarán a las mismas. Una mala elección podría tener fuertes implicaciones de tiempo, dinero y recursos malgastados.

Para evitar esto se ha llevado a cabo un análisis sobre las comunicaciones en tiempo real y se ha elegido WebRTC pues parecía ser la más adecuada para el proyecto. Teniendo en cuenta los resultados mostrados en la sección ?? esta elección (y su posterior implementación derivada) ha resultado todo un acierto.

Una vez realizado el análisis basado en los avances realizados por la comunidad científica hasta la actualidad y el diseño de la arquitectura de módulos de la plataforma, llega el momento de la implementación. El resultado final ha sido un sistema distribuido para asistir en las labores de ayuda de un equipo de emergencias desplegado en el lugar de un accidente mediante la aplicación de realidad mixta y telemedicina, cumpliendo así el objetivo general del proyecto.

El sistema finalizado es completamente funcional y permite a dicho equipo de emergencias recibir indicaciones de un especialista remoto mediante audio, vídeo y marcadores con baja latencia y buen rendimiento.

Además, también se ha logrado integrar dispositivos médicos en el sistema y distribuir los datos procedentes de los mismos en tiempo real a cada uno de los participantes de la videoconferencia con el fin de proporcionar toda la información posible, lo que mejora la precisión de los diagnósticos y el tiempo de respuesta.

6.4. LÍNEAS DE TRABAJO FUTURO

A continuación se exponen algunas líneas de trabajo futuro que se van a tratar durante los próximos meses, ya que el proyecto 5GCLM se continuará desarrollando en los siguientes meses.

- **Escritura de un artículo JCR.** En el capítulo de introducción de este mismo documento se menciona que el proyecto se enmarca dentro de la segunda convocatoria de subvenciones a proyectos piloto de tecnologías 5G. Uno de los requisitos planteados en la difusión del proyecto es la publicación de un *paper* sobre este trabajo en una revista Q1/Q2, el cual se encuentra ahora mismo en proceso de escritura.
- **Nuevos casos de uso con drones.** El uso de drones y otros vehículos no tripulados para hacer llegar el equipamiento a usuarios sin conocimientos médicos en situaciones de emergencia se considera muy interesante (aunque no era un requisito obligatorio en la ejecución del proyecto), por lo que se procederá a evaluar su viabilidad en los próximos meses con vistas a la realización de un prototipo.
- **Estudio de otros cascos de RM.** Estudio de viabilidad de otro hardware de realidad mixta de menor precio y mejor soporte técnico. Quest Pro² es un nuevo casco de realidad aumentada y virtual fabricado por Meta y de un coste mucho menor en comparación con el de HoloLens 2, por lo que se está planteando hacer una versión de la aplicación Health-5G_Holo para este ya que muchas tecnologías de las utilizadas en el desarrollo de este proyecto son compatibles.
- **Nuevos dominios de aplicación.** Aunque la temática de este trabajo se haya centrado en torno a la medicina de emergencias, los resultados del mismo podrían ser aplicables con modificaciones menores a un entorno industrial. Un ejemplo podría ser en la reparación de vehículos, donde un ingeniero de la marca podría guiar a un mecánico en ciertas operaciones complejas.

²<https://www.meta.com/es/quest/quest-pro/>

ANEXOS

Documentación del proyecto

A continuación se incluye la documentación del proyecto pues se considera útil que el lector la tenga a mano a modo de material de consulta. Se ha generado utilizando la herramienta Doxygen¹ a partir de comentarios en el código.

Sí bien este anexo es una versión reducida de la documentación para evitar que la memoria de este trabajo de fin de máster sea demasiado voluminosa, cualquier persona interesada en consultar una versión más extensa de la misma podría generarla a conveniencia en base a los diversos archivos *Doxyfile* que se incluyen junto al código fuente.

¹<https://doxygen.nl/>

Health-5G
Prototype 4

Generado por Doxygen 1.9.5

	I
1 Índice jerárquico	1
1.1 Jerarquía de la clase	1
2 Índice de clases	3
2.1 Lista de clases	3
3 Documentación de las clases	5
3.1 Referencia de la Clase ArrowAnimation	5
3.1.1 Descripción detallada	5
3.2 Referencia de la Clase AxisGizmoManager	6
3.2.1 Descripción detallada	6
3.2.2 Documentación de las funciones miembro	6
3.3 Referencia de la Clase AxisGizmoMessage	7
3.3.1 Descripción detallada	7
3.4 Referencia de la Clase AxisGizmoMessageReceivedEvent	7
3.4.1 Descripción detallada	7
3.5 Referencia de la Clase CallButtonClickedEvent	8
3.5.1 Descripción detallada	8
3.6 Referencia de la Clase CallEndedEvent	8
3.6.1 Descripción detallada	9
3.7 Referencia de la Clase CallFailedEvent	9
3.7.1 Descripción detallada	9
3.8 Referencia de la Clase CallStartedEvent	9
3.8.1 Descripción detallada	10
3.9 Referencia de la Clase CleanUpManager	10
3.9.1 Descripción detallada	10
3.10 Referencia de la Clase ClickedVideoMessage	11
3.10.1 Descripción detallada	11
3.11 Referencia de la Clase ClickedVideoMessageReceivedEvent	11
3.11.1 Descripción detallada	12
3.12 Referencia de la Clase ConfigurableMonoBehaviour	12
3.12.1 Descripción detallada	13
3.12.2 Documentación de las funciones miembro	13
3.13 Referencia de la Clase ConnectionManager	14
3.13.1 Descripción detallada	16
3.13.2 Documentación de las funciones miembro	16
3.14 Referencia de la Clase DataChannelAddedEvent	18
3.14.1 Descripción detallada	18
3.15 Referencia de la Clase DemoSignalingServer	18
3.15.1 Descripción detallada	19
3.16 Referencia de la Clase FacePlayerAnimation	20
3.16.1 Descripción detallada	20
3.17 Referencia de la Clase HandMenuFunctions	20

II

3.17.1 Descripción detallada	22
3.18 Referencia de la Clase ImageDownloaderManager	22
3.18.1 Descripción detallada	23
3.19 Referencia de la Clase ImageDownloaderManagerCanvas	23
3.19.1 Descripción detallada	24
3.20 Referencia de la Clase ImageDownloadManagerHub	24
3.20.1 Descripción detallada	25
3.20.2 Documentación de las funciones miembro	25
3.21 Referencia de la Clase InitialPositionAndScale	25
3.21.1 Descripción detallada	26
3.22 Referencia de la Clase InteractableOverlay	26
3.22.1 Descripción detallada	27
3.22.2 Documentación de las funciones miembro	28
3.23 Referencia de la Clase LoggerManager	28
3.23.1 Descripción detallada	29
3.24 Referencia de la Clase MessageManager	29
3.24.1 Descripción detallada	30
3.24.2 Documentación de las funciones miembro	30
3.25 Referencia de la Clase NodeDssSignalerConfig	32
3.25.1 Descripción detallada	33
3.25.2 Documentación de las funciones miembro	33
3.26 Referencia de la Clase PeerConnectionConfig	34
3.26.1 Descripción detallada	34
3.26.2 Documentación de las funciones miembro	35
3.27 Referencia de la Clase RepetitiveButton	35
3.27.1 Descripción detallada	36
3.27.2 Documentación de las funciones miembro	36
3.28 Referencia de la Clase RepetitiveButtonClickedEvent	37
3.28.1 Descripción detallada	37
3.29 Referencia de la Clase ResetVideoPlayer	37
3.29.1 Descripción detallada	38
3.30 Referencia de la Clase ResizeBoxCollider2dToTransform	38
3.30.1 Descripción detallada	38
3.31 Referencia de la Clase ShapeIdentifier	39
3.31.1 Descripción detallada	39
3.32 Referencia de la Clase ShapeManager	39
3.32.1 Descripción detallada	41
3.32.2 Documentación de las funciones miembro	41
3.33 Referencia de la Clase ShapeModifiersMessage	43
3.33.1 Descripción detallada	43
3.34 Referencia de la Clase ShapeModifiersMessageReceivedEvent	44
3.34.1 Descripción detallada	44

1 Índice jerárquico	1
3.35 Referencia de la Clase ShapeVoiceCommands	44
3.35.1 Descripción detallada	45
3.36 Referencia de la Clase SpaceObjectsManager	45
3.36.1 Descripción detallada	47
3.36.2 Documentación de las funciones miembro	47
3.37 Referencia de la Clase UIManager	48
3.37.1 Descripción detallada	51
3.37.2 Documentación de las funciones miembro	51
3.38 Referencia de la Clase VideoClickedEvent	52
3.38.1 Descripción detallada	53
3.39 Referencia de la Clase VideoDraggedEvent	53
3.39.1 Descripción detallada	54
3.40 Referencia de la Clase VideoUppedEvent	54
3.40.1 Descripción detallada	54
Índice alfabético	55

1. Índice jerárquico

1.1. Jerarquía de la clase

Esta lista de herencias esta ordenada aproximadamente por orden alfabético:

AxisGizmoMessage	7
ClickedVideoMessage	11
IPointerDownHandler	
RepetitiveButton	35
IPointerUpHandler	
RepetitiveButton	35
MonoBehaviour	
ArrowAnimation	5
AxisGizmoManager	6
CleanUpManager	10
ConfigurableMonoBehaviour	12
ImageDownloadManagerHub	24
InteractableOverlay	26
NodeDssSignalerConfig	32
PeerConnectionConfig	34
SpaceObjectsManager	45

UIManager	48
ConnectionManager	14
DemoSignalingServer	18
FacePlayerAnimation	20
FacePlayerAnimation	20
HandMenuFunctions	20
ImageDownloaderManager	22
ImageDownloaderManagerCanvas	23
InitialPositionAndScale	25
LoggerManager	28
MessageManager	29
RepetitiveButton	35
ResetVideoPlayer	37
ResizeBoxCollider2dToTransform	38
ShapeIdentifier	39
ShapeManager	39
ShapeVoiceCommands	44
ShapeModifiersMessage	43
UnityEvent	
AxisGizmoMessageReceivedEvent	7
CallButtonClickedEvent	8
CallEndedEvent	8
CallFailedEvent	9
CallStartedEvent	9
ClickedVideoMessageReceivedEvent	11
DataChannelAddedEvent	18
RepetitiveButtonClickedEvent	37
ShapeModifiersMessageReceivedEvent	44
VideoClickedEvent	52
VideoDraggedEvent	53
VideoUppedEvent	54

2 Índice de clases	3
--------------------	---

2. Índice de clases

2.1. Lista de clases

Lista de las clases, estructuras, uniones e interfaces con una breve descripción:

ArrowAnimation		
Animacion de giro del marcador flecha.		5
AxisGizmoManager		
Envia la rotacion de la cabeza del usuario o gira el gizmo, segun corresponda.		6
AxisGizmoMessage		
Mensaje de la rotacion de la cabeza del usuario.		7
AxisGizmoMessageReceivedEvent		
El evento AxisGizmoMessageReceivedEvent se invoca cuando llega un mensaje de tipo AxisGizmoMessage .		7
CallButtonClickedEvent		
Evento que se lanza cuando se pulsa el boton de iniciar llamada y no hay llamada en proceso.		8
CallEndedEvent		
Evento que se lanza cuando se ha terminado una llamada.		8
CallFailedEvent		
Evento que se lanza cuando una llamada falla al intentar iniciarla.		9
CallStartedEvent		
Evento que se lanza cuando se ha iniciado de forma exitosa una llamada.		9
CleanUpManager		
Desactiva los objetos de la escena antes de salir de la aplicacion.		10
ClickedVideoMessage		
Mensaje que se lanza cuando se interactua con el video proveniente de HoloLens.		11
ClickedVideoMessageReceivedEvent		
El evento ClickedVideoMessageReceivedEvent se invoca cuando llega un mensaje de tipo ClickedVideoMessage .		11
ConfigurableMonoBehaviour		
Clase de la que deberan heredar las que quieran guardar variables en el archivo de configuracion.		12
ConnectionManager		
La clase singleton ConnectionManager encapsula y simplifica el funcionamiento de MixedReality WebRTC para permitir su utilizacion de forma sencilla en cualquier aplicacion. Es necesario que en la escena donde se use este script haya tambien presente un objeto del tipo PeerConnection .		14
DataChannelAddedEvent		
Evento que indica que se ha creado un nuevo canal de datos.		18
DemoSignalingServer		
Servidor de signaling para demos offline.		18
FacePlayerAnimation		
El objeto que lleve este script siempre mirara a la camara principal.		20

HandMenuFunctions	20
Clase que encapsula las funciones del menu mano.	
ImageDownloaderManager	22
Descarga imagenes y las renderiza sobre un objeto de tipo Material.	
ImageDownloaderManagerCanvas	23
Descarga imagenes y las renderiza sobre un objeto de tipo Image.	
ImageDownloadManagerHub	24
Configura multiples objetos de tipo ImageDownloaderManager o ImageDownloaderManagerCanvas a la vez.	
InitialPositionAndScale	25
Guarda la posicion, escala y rotacion de un objeto y permite resetearlo.	
InteractableOverlay	26
Este script recoge eventos de raton sobre un video y los convierte en una serie de coordenadas normalizadas.	
LoggerManager	28
Apaga el log de Unity en la version compilada de la aplicacion.	
MessageManager	29
Facilita el envio y la recepcion de mensajes a traves de WebRTC.	
NodeDssSignalerConfig	32
Configura un objeto de tipo NodeDssSignaler.	
PeerConnectionConfig	34
Configura un objeto de tipo PeerConnection.	
RepetitiveButton	35
Script que permite que un boton actue repetidamente mientras se tiene pulsado.	
RepetitiveButtonClickedEvent	37
Evento que se lanzara mientras se tenga el boton pulsado.	
ResetVideoPlayer	37
Resetea varios Renderer con video YUV a un color por defecto.	
ResizeBoxCollider2dToTransform	38
Modifica un collider para que encaje con una transformacion.	
ShapeIdentifier	39
Identifica este marcador para saber si es una flecha, una caja, una cruz o un dibujo.	
ShapeManager	39
Genera marcadores y ofrece utilidades para borrarlos y seleccionarlos.	
ShapeModifiersMessage	43
Mensaje que se envia al utilizar las herramientas de modificacion de marcadores.	
ShapeModifiersMessageReceivedEvent	44
El evento ShapeModifiersMessageReceivedEvent se invoca cuando llega un mensaje de tipo ShapeModifiersMessage .	
ShapeVoiceCommands	44
Ofrece comandos de voz para marcadores 3D.	

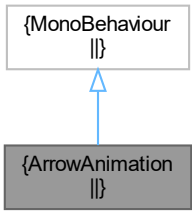
3 Documentación de las clases	5
SpaceObjectsManager	
Manager que se encarga del emplazamiento de los marcadores en el espacio.	45
UIManager	
Manager de la UI de la app de escritorio.	48
VideoClickedEvent	
Evento que se lanza cuando se hace click en el video.	52
VideoDraggedEvent	
Evento que se lanza cuando se arrastra en el video.	53
VideoUppedEvent	
Evento que se lanza cuando se deja de hacer click en el video.	54

3. Documentación de las clases

3.1. Referencia de la Clase ArrowAnimation

Animacion de giro del marcador flecha.

Diagrama de herencias de ArrowAnimation



3.1.1. Descripción detallada

Animacion de giro del marcador flecha.

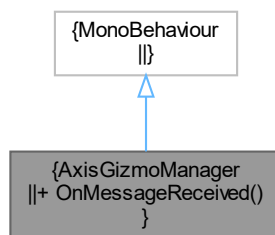
La documentación para esta clase fue generada a partir del siguiente fichero:

- Health-5G_Holo/Assets/Scripts/Animations/ArrowAnimation.cs

3.2. Referencia de la Clase AxisGizmoManager

Envía la rotacion de la cabeza del usuario o gira el gizmo, segun corresponda.

Diagrama de herencias de AxisGizmoManager



Métodos públicos

- void `OnMessageReceived` (`AxisGizmoMessage` msg)
Recibe un mensaje `AxisGizmoMessage` y lo pone en una cola.

3.2.1. Descripción detallada

Envía la rotacion de la cabeza del usuario o gira el gizmo, segun corresponda.

3.2.2. Documentación de las funciones miembro

3.2.2.1. OnMessageReceived() void `AxisGizmoManager.OnMessageReceived` (
`AxisGizmoMessage` msg)

Recibe un mensaje `AxisGizmoMessage` y lo pone en una cola.

Parámetros

<code>msg</code>	El mensaje de tipo <code>AxisGizmoMessage</code> .
------------------	--

La documentación para esta clase fue generada a partir del siguiente fichero:

- Shared/Managers/AxisGizmoManager.cs

3.3 Referencia de la Clase AxisGizmoMessage

7

3.3. Referencia de la Clase AxisGizmoMessage

Mensaje de la rotacion de la cabeza del usuario.

Atributos públicos

- float **x**
Componente x del vector de rotacion.
- float **y**
Componente y del vector de rotacion.
- float **z**
Componente z del vector de rotacion.

3.3.1. Descripción detallada

Mensaje de la rotacion de la cabeza del usuario.

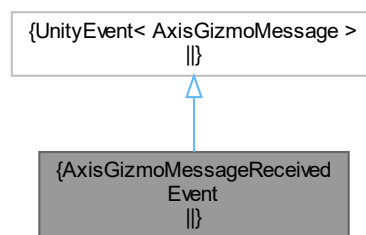
La documentación para esta clase fue generada a partir del siguiente fichero:

- Shared/Messages/AxisGizmoMessage.cs

3.4. Referencia de la Clase AxisGizmoMessageReceivedEvent

El evento [AxisGizmoMessageReceivedEvent](#) se invoca cuando llega un mensaje de tipo [AxisGizmoMessage](#).

Diagrama de herencias de AxisGizmoMessageReceivedEvent



3.4.1. Descripción detallada

El evento [AxisGizmoMessageReceivedEvent](#) se invoca cuando llega un mensaje de tipo [AxisGizmoMessage](#).

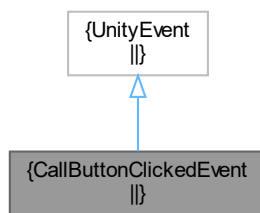
La documentación para esta clase fue generada a partir del siguiente fichero:

- Shared/Managers/MessageManager.cs

3.5. Referencia de la Clase CallButtonClickedEvent

Evento que se lanza cuando se pulsa el boton de iniciar llamada y no hay llamada en proceso.

Diagrama de herencias de CallButtonClickedEvent



3.5.1. Descripción detallada

Evento que se lanza cuando se pulsa el boton de iniciar llamada y no hay llamada en proceso.

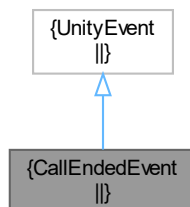
La documentación para esta clase fue generada a partir del siguiente fichero:

- Shared/WebRTC/ConnectionManager.cs

3.6. Referencia de la Clase CallEndedEvent

Evento que se lanza cuando se ha terminado una llamada.

Diagrama de herencias de CallEndedEvent



3.7 Referencia de la Clase CallFailedEvent

9

3.6.1. Descripción detallada

Evento que se lanza cuando se ha terminado una llamada.

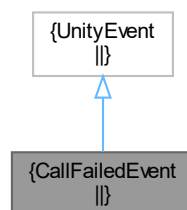
La documentación para esta clase fue generada a partir del siguiente fichero:

- Shared/WebRTC/ConnectionManager.cs

3.7. Referencia de la Clase CallFailedEvent

Evento que se lanza cuando una llamada falla al intentar iniciarla.

Diagrama de herencias de CallFailedEvent



3.7.1. Descripción detallada

Evento que se lanza cuando una llamada falla al intentar iniciarla.

La documentación para esta clase fue generada a partir del siguiente fichero:

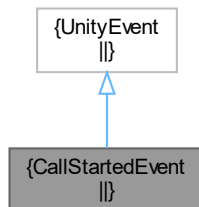
- Shared/WebRTC/ConnectionManager.cs

3.8. Referencia de la Clase CallStartedEvent

Evento que se lanza cuando se ha iniciado de forma exitosa una llamada.

10

Diagrama de herencias de CallStartedEvent



3.8.1. Descripción detallada

Evento que se lanza cuando se ha iniciado de forma exitosa una llamada.

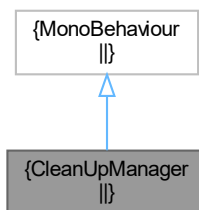
La documentación para esta clase fue generada a partir del siguiente fichero:

- Shared/WebRTC/ConnectionManager.cs

3.9. Referencia de la Clase CleanUpManager

Desactiva los objetos de la escena antes de salir de la aplicacion.

Diagrama de herencias de CleanUpManager



3.9.1. Descripción detallada

Desactiva los objetos de la escena antes de salir de la aplicacion.

La documentación para esta clase fue generada a partir del siguiente fichero:

- Shared/Managers/CleanUpManager.cs

3.10 Referencia de la Clase ClickedVideoMessage

11

3.10. Referencia de la Clase ClickedVideoMessage

Mensaje que se lanza cuando se interactua con el video proveniente de HoloLens.

Atributos públicos

- string **videoid**
ID del video sobre el que se ha hecho click.
- int **shapeld**
ID de la forma que se quiere poner Arrow = 0, Box = 1, Cross = 2, Line = 3
- float **x**
Coordenada 'x' en el intervalo [0, 1]
- float **y**
Coordenada 'y' en el intervalo [0, 1]
- float **noHitObjectDistance**
Distancia por defecto a la que colocar el marcador si el raycasting no surte efecto
- int **mouseActionId**
Acción del mouse Down = 0, Drag = 1, Up = 2

3.10.1. Descripción detallada

Mensaje que se lanza cuando se interactua con el video proveniente de HoloLens.

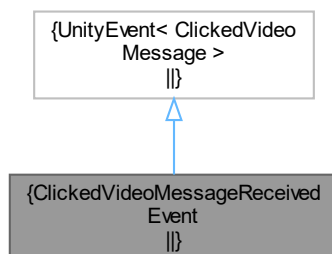
La documentación para esta clase fue generada a partir del siguiente fichero:

- Shared/Messages/ClickedVideoMessage.cs

3.11. Referencia de la Clase ClickedVideoMessageReceivedEvent

El evento [ClickedVideoMessageReceivedEvent](#) se invoca cuando llega un mensaje de tipo [ClickedVideoMessage](#).

Diagrama de herencias de ClickedVideoMessageReceivedEvent



3.11.1. Descripción detallada

El evento [ClickedVideoMessageReceivedEvent](#) se invoca cuando llega un mensaje de tipo [ClickedVideoMessage](#).

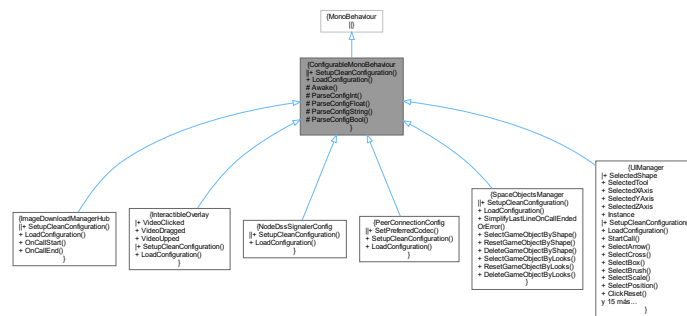
La documentación para esta clase fue generada a partir del siguiente fichero:

- Shared/Managers/MessageManager.cs

3.12. Referencia de la Clase ConfigurableMonoBehaviour

Clase de la que deberán heredar las que quieran guardar variables en el archivo de configuracion.

Diagrama de herencias de ConfigurableMonoBehaviour



Métodos públicos

- abstract void [SetupCleanConfiguration](#) (Configuration cfg)
Carga la configuracion por defecto en un objeto Configuration.
- abstract void [LoadConfiguration](#) (Configuration cfg)
Carga la configuracion desde un archivo config.cfg.

Métodos protegidos

- virtual void **Awake** ()
Previene que las clases herederas usen Awake para evitar problemas al cargar la configuracion.
- void [ParseConfigInt](#) (Setting setting, ref int defaultParameter)
Parsea una variable desde un archivo de configuracion.
- void [ParseConfigFloat](#) (Setting setting, ref float defaultParameter)
Parsea una variable desde un archivo de configuracion.
- void [ParseConfigString](#) (Setting setting, ref string defaultParameter)
Parsea una variable desde un archivo de configuracion.
- void [ParseConfigBool](#) (Setting setting, ref bool defaultParameter)
Parsea una variable desde un archivo de configuracion.

3.12 Referencia de la Clase ConfigurableMonoBehaviour

13

3.12.1. Descripción detallada

Clase de la que deberán heredar las que quieran guardar variables en el archivo de configuración.

3.12.2. Documentación de las funciones miembro

3.12.2.1. LoadConfiguration() `abstract void ConfigurableMonoBehaviour.LoadConfiguration (Configuration cfg) [pure virtual]`

Carga la configuración desde un archivo config.cfg.

Parámetros

<i>cfg</i>	El objeto Configuration incluye la configuración de la aplicación.
------------	--

Implementado en [InteractiveOverlay](#), [UIManager](#), [SpaceObjectsManager](#), [NodeDssSignalerConfig](#), [PeerConnectionConfig](#) y [ImageDownloadManagerHub](#).

3.12.2.2. ParseConfigBool() `void ConfigurableMonoBehaviour.ParseConfigBool (Setting setting, ref bool defaultParameter) [protected]`

Parsea una variable desde un archivo de configuración.

Parámetros

<i>setting</i>	El parámetro Setting indica la configuración de una variable.
<i>defaultParameter</i>	Valor por defecto del parámetro a parsear.

3.12.2.3. ParseConfigFloat() `void ConfigurableMonoBehaviour.ParseConfigFloat (Setting setting, ref float defaultParameter) [protected]`

Parsea una variable desde un archivo de configuración.

Parámetros

<i>setting</i>	El parámetro Setting indica la configuración de una variable.
<i>defaultParameter</i>	Valor por defecto del parámetro a parsear.

14

3.12.2.4. ParseConfigInt() `void ConfigurableMonoBehaviour.ParseConfigInt (`
 `Setting setting,`
 `ref int defaultParameter) [protected]`

Parsea una variable desde un archivo de configuracion.

Parámetros

<i>setting</i>	El parametro Setting indica la configuracion de una variable.
<i>defaultParameter</i>	Valor por defecto del parametro a parsear.

3.12.2.5. ParseConfigString() `void ConfigurableMonoBehaviour.ParseConfigString (`
 `Setting setting,`
 `ref string defaultParameter) [protected]`

Parsea una variable desde un archivo de configuracion.

Parámetros

<i>setting</i>	El parametro Setting indica la configuracion de una variable.
<i>defaultParameter</i>	Valor por defecto del parametro a parsear.

3.12.2.6. SetupCleanConfiguration() `abstract void ConfigurableMonoBehaviour.SetupCleanConfiguration`
 `(`
 `Configuration cfg) [pure virtual]`

Carga la configuracion por defecto en un objeto Configuration.

Parámetros

<i>cfg</i>	El objeto Configuration incluye la configuracion de la aplicacion.
------------	--

Implementado en [InteractableOverlay](#), [UIManager](#), [SpaceObjectsManager](#), [NodeDssSignalerConfig](#), [PeerConnectionConfig](#) y [ImageDownloadManagerHub](#).

La documentación para esta clase fue generada a partir del siguiente fichero:

- Shared/Configuration/ConfigurableMonoBehaviour.cs

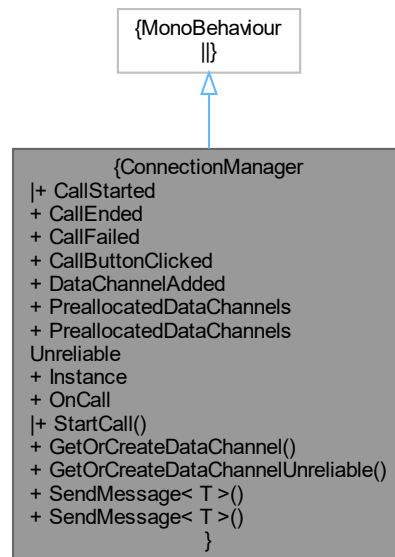
3.13. Referencia de la Clase ConnectionManager

La clase singleton [ConnectionManager](#) encapsula y simplifica el funcionamiento de MixedReality WebRTC para permitir su utilizacion de forma sencilla en cualquier aplicacion. Es necesario que en la escena donde se use este script haya tambien presente un objeto del tipo `PeerConnection`.

3.13 Referencia de la Clase ConnectionManager

15

Diagrama de herencias de ConnectionManager



Métodos públicos

- `void StartCall ()`
Inicia o termina la llamada, según corresponda.
- `Microsoft.MixedReality.WebRTC.DataChannel GetOrCreateDataChannel (string channelLabel)`
Devuelve un canal de datos a partir de su label (o lo crea si no existe).
- `Microsoft.MixedReality.WebRTC.DataChannel GetOrCreateDataChannelUnreliable (string channelLabel)`
Devuelve un canal de datos no fiable a partir de su label (o lo crea si no existe).
- `bool SendMessage< T > (T msg, string channelLabel)`
Envía un mensaje por el canal de datos correspondiente.
- `bool SendMessage< T > (T msg, Microsoft.MixedReality.WebRTC.DataChannel channel)`
Envía un mensaje por el canal de datos correspondiente.

Propiedades

- `CallStartedEvent CallStarted [get]`
*Retorna el evento **CallStarted***
- `CallEndedEvent CallEnded [get]`
*Retorna el evento **CallEnded***
- `CallFailedEvent CallFailed [get]`
*Retorna el evento **CallFailedEvent***

- **CallButtonClickedEvent CallButtonClicked** [get]
Retorna el evento [CallButtonClickedEvent](#)
- **DataChannelAddedEvent DataChannelAdded** [get]
Retorna el evento [DataChannelAddedEvent](#).
- **string[] PreallocatedDataChannels** [get, set]
Aunque esta clase de utilidades soporta la creacion de canales de datos en cualquier momento de la llamada, esta propiedad permite que una serie de canales por defecto sean iniciados al empezar la llamada. Es buena practica que los canales que se vayan a usar si o si se pongan en esta propiedad para que se creen al momento de iniciar la llamada.
- **string[] PreallocatedDataChannelsUnreliable** [get, set]
Aunque esta clase de utilidades soporta la creacion de canales de datos en cualquier momento de la llamada, esta propiedad permite que una serie de canales por defecto sean iniciados al empezar la llamada. Es buena practica que los canales que se vayan a usar si o si se pongan en esta propiedad para que se creen al momento de iniciar la llamada.
- **static ConnectionManager Instance** [get]
Gets the Instance Instancia activa de [ConnectionManager](#).
- **bool OnCall** [get, set]
Retorna la variable que indica si nos encontramos dentro de una llamada.

3.13.1. Descripción detallada

La clase singleton [ConnectionManager](#) encapsula y simplifica el funcionamiento de MixedReality WebRTC para permitir su utilizacion de forma sencilla en cualquier aplicacion. Es necesario que en la escena donde se use este script haya tambien presente un objeto del tipo [PeerConnection](#).

3.13.2. Documentación de las funciones miembro

3.13.2.1. GetOrCreateDataChannel() [Microsoft.MixedReality.WebRTC.DataChannel](#) [ConnectionManager](#).↔
`GetOrCreateDataChannel (`
 `string channelLabel)`

Devuelve un canal de datos a partir de su label (o lo crea si no existe).

Parámetros

<code>channelLabel</code>	El nombre del canal de datos.
---------------------------	-------------------------------

Devuelve

El canal [Microsoft.MixedReality.WebRTC.DataChannel](#).

3.13.2.2. GetOrCreateDataChannelUnreliable() [Microsoft.MixedReality.WebRTC.DataChannel](#) [ConnectionManager](#).↔
`Manager.GetOrCreateDataChannelUnreliable (`
 `string channelLabel)`

Devuelve un canal de datos no fiable a partir de su label (o lo crea si no existe).

3.13 Referencia de la Clase ConnectionManager**17****Parámetros**

<i>channelLabel</i>	El nombre del canal de datos.
---------------------	-------------------------------

Devuelve

El canal `Microsoft.MixedReality.WebRTC.DataChannel`.

3.13.2.3. `SendMessage< T >()` [1/2] `bool ConnectionManager.SendMessage< T > (`
`T msg,`
`Microsoft.MixedReality.WebRTC.DataChannel channel)`

Envía un mensaje por el canal de datos correspondiente.

Parámetros del template

<i>T</i>	Tipo del mensaje a enviar.
----------	----------------------------

Parámetros

<i>msg</i>	Objeto que se va a enviar.
<i>channel</i>	Referencia del canal de datos.

Devuelve

True si se ha enviado correctamente, false si no.

3.13.2.4. `SendMessage< T >()` [2/2] `bool ConnectionManager.SendMessage< T > (`
`T msg,`
`string channelLabel)`

Envía un mensaje por el canal de datos correspondiente.

Parámetros del template

<i>T</i>	Tipo del mensaje a enviar.
----------	----------------------------

Parámetros

<i>msg</i>	Objeto que se va a enviar.
<i>channelLabel</i>	Nombre del canal de datos.

18

Devuelve

True si se ha enviado correctamente, false si no.

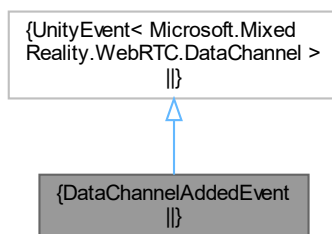
La documentación para esta clase fue generada a partir del siguiente fichero:

- Shared/WebRTC/ConnectionManager.cs

3.14. Referencia de la Clase DataChannelAddedEvent

Evento que indica que se ha creado un nuevo canal de datos.

Diagrama de herencias de DataChannelAddedEvent



3.14.1. Descripción detallada

Evento que indica que se ha creado un nuevo canal de datos.

La documentación para esta clase fue generada a partir del siguiente fichero:

- Shared/WebRTC/ConnectionManager.cs

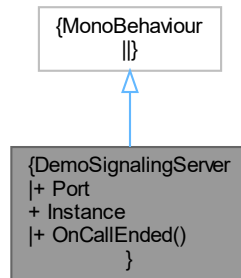
3.15. Referencia de la Clase DemoSignalingServer

Servidor de signaling para demos offline.

3.15 Referencia de la Clase DemoSignalingServer

19

Diagrama de herencias de DemoSignalingServer



Métodos públicos

- `void OnCallEnded ()`
Se llamara cuando acabe la llamada.

Propiedades

- `int Port [get]`
Retorna el puerto del servidor.
- `static DemoSignalingServer Instance [get]`
Instancia del servidor de signaling.

3.15.1. Descripción detallada

Servidor de signaling para demos offline.

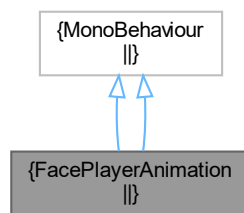
La documentación para esta clase fue generada a partir del siguiente fichero:

- `Shared/WebRTC/DemoSignalingServer.cs`

3.16. Referencia de la Clase FacePlayerAnimation

El objeto que lleve este script siempre mirara a la camara principal.

Diagrama de herencias de FacePlayerAnimation



3.16.1. Descripción detallada

El objeto que lleve este script siempre mirara a la camara principal.

La documentación para esta clase fue generada a partir de los siguientes ficheros:

- Health-5G_Holo/Assets/Scripts/Animations/FacePlayerAnimation.cs
- Health-5G_Desktop/Assets/Scripts/Animations/FacePlayerAnimation.cs

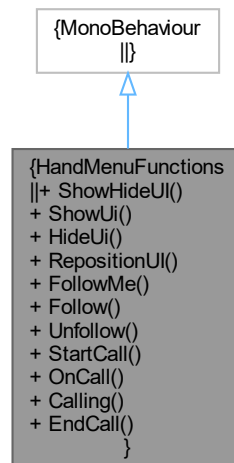
3.17. Referencia de la Clase HandMenuFunctions

Clase que encapsula las funciones del menu mano.

3.17 Referencia de la Clase HandMenuFunctions

21

Diagrama de herencias de HandMenuFunctions



Métodos públicos

- void **ShowHideUI** ()
Muestra u oculta la UI.
- void **ShowUi** ()
Muestra la UI.
- void **HideUi** ()
Oculto la UI.
- void **RepositionUI** ()
Reposiciona la UI.
- void **FollowMe** ()
Alterna la UI entre los estados de seguir y no seguir al usuario.
- void **Follow** ()
Hace que la UI siga al usuario.
- void **Unfollow** ()
Hace que la UI deje de seguir al usuario.
- void **StartCall** ()
Inicia una llamada.
- void **OnCall** ()
Cambia la textura del boton de llamada a la version de "en llamada".
- void **Calling** ()
Cambia la textura del boton de llamada a la version de "llamando".
- void **EndCall** ()
Cambia la textura del boton de llamada a la version de "iniciar llamada".

3.17.1. Descripción detallada

Clase que encapsula las funciones del menu mano.

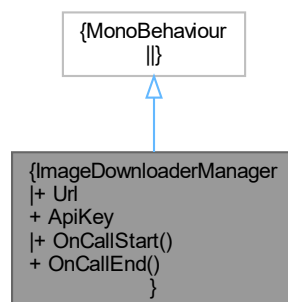
La documentación para esta clase fue generada a partir del siguiente fichero:

- Health-5G_Holo/Assets/Scripts/UI/HandMenuFunctions.cs

3.18. Referencia de la Clase ImageDownloaderManager

Descarga imagenes y las renderiza sobre un objeto de tipo Material.

Diagrama de herencias de ImageDownloaderManager



Métodos públicos

- void **OnCallStart** ()
Metodo que sera llamado cuando se inicie la llamada.
- void **OnCallEnd** ()
Metodo que sera llamado cuando se acabe la llamada.

Propiedades

- string **Url** [get, set]
Devuelve o cambia la URL de descarga de imagenes.
- string **ApiKey** [get, set]
Devuelve o cambia la clave de API.

3.19 Referencia de la Clase ImageDownloaderManagerCanvas

23

3.18.1. Descripción detallada

Descarga imagenes y las renderiza sobre un objeto de tipo Material.

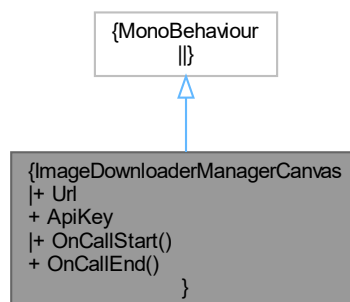
La documentación para esta clase fue generada a partir del siguiente fichero:

- Shared/Managers/ImageDownloaderManager.cs

3.19. Referencia de la Clase ImageDownloaderManagerCanvas

Descarga imagenes y las renderiza sobre un objeto de tipo Image.

Diagrama de herencias de ImageDownloaderManagerCanvas



Métodos públicos

- void **OnCallStart** ()
Metodo que sera llamado cuando se inicie la llamada.
- void **OnCallEnd** ()
Metodo que sera llamado cuando se acabe la llamada.

Propiedades

- string **Url** [get, set]
Devuelve o cambia la URL de descarga de imagenes.
- string **ApiKey** [get, set]
Devuelve o cambia la clave de API.

3.19.1. Descripción detallada

Descarga imagenes y las renderiza sobre un objeto de tipo Image.

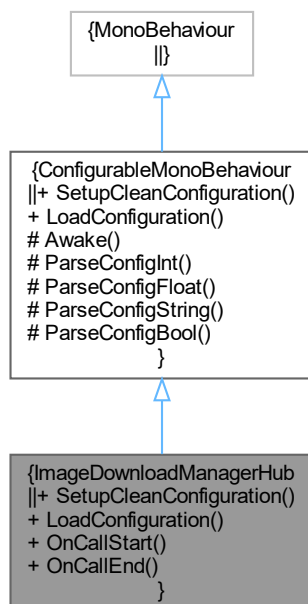
La documentación para esta clase fue generada a partir del siguiente fichero:

- Shared/Managers/ImageDownloaderManagerCanvas.cs

3.20. Referencia de la Clase ImageDownloadManagerHub

Configura multiples objetos de tipo [ImageDownloaderManager](#) o [ImageDownloaderManagerCanvas](#) a la vez.

Diagrama de herencias de ImageDownloadManagerHub



Métodos públicos

- override void [SetupCleanConfiguration](#) (Configuration cfg)
Carga la configuracion por defecto en un objeto Configuration.
- override void [LoadConfiguration](#) (Configuration cfg)
Carga la configuracion desde un archivo config.cfg.
- void **OnCallStart** ()
Metodo que sera llamado cuando se inicie la llamada.
- void **OnCallEnd** ()
Metodo que sera llamado cuando se acabe la llamada.

3.21 Referencia de la Clase InitialPositionAndScale

25

Otros miembros heredados

3.20.1. Descripción detallada

Configura multiples objetos de tipo [ImageDownloaderManager](#) o [ImageDownloaderManagerCanvas](#) a la vez.

3.20.2. Documentación de las funciones miembro

3.20.2.1. LoadConfiguration() `override void ImageDownloadManagerHub.LoadConfiguration (Configuration cfg) [virtual]`

Carga la configuracion desde un archivo config.cfg.

Parámetros

<i>cfg</i>	El objeto Configuration incluye la configuracion de la aplicacion.
------------	--

Implementa [ConfigurableMonoBehaviour](#).

3.20.2.2. SetupCleanConfiguration() `override void ImageDownloadManagerHub.SetupCleanConfiguration (Configuration cfg) [virtual]`

Carga la configuracion por defecto en un objeto Configuration.

Parámetros

<i>cfg</i>	El objeto Configuration incluye la configuracion de la aplicacion.
------------	--

Implementa [ConfigurableMonoBehaviour](#).

La documentación para esta clase fue generada a partir del siguiente fichero:

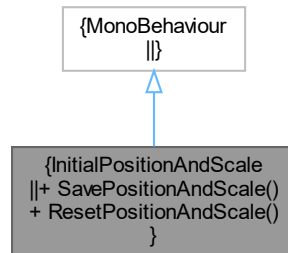
- Shared/Managers/ImageDownloadManagerHub.cs

3.21. Referencia de la Clase InitialPositionAndScale

Guarda la posicion, escala y rotacion de un objeto y permite resetearlo.

26

Diagrama de herencias de InitialPositionAndScale



Métodos públicos

- void **SavePositionAndScale** ()
Guarda la posicion, escala y rotacion que tenga en este momento el objeto.
- void **ResetPositionAndScale** ()
Resetea la posicion, escala y rotacion a la que este guardada.

3.21.1. Descripción detallada

Guarda la posicion, escala y rotacion de un objeto y permite resetearlo.

La documentación para esta clase fue generada a partir del siguiente fichero:

- Health-5G_Holo/Assets/Scripts/GameObjects/InitialPositionAndScale.cs

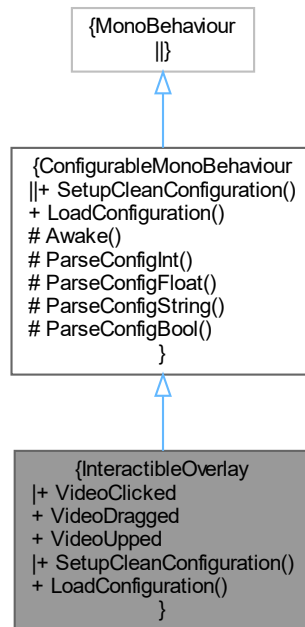
3.22. Referencia de la Clase InteractableOverlay

Este script recoge eventos de raton sobre un video y los convierte en una serie de coordenadas normalizadas.

3.22 Referencia de la Clase InteractableOverlay

27

Diagrama de herencias de InteractableOverlay



Métodos públicos

- override void [SetupCleanConfiguration](#) (Configuration cfg)
Carga la configuracion por defecto en un objeto Configuration.
- override void [LoadConfiguration](#) (Configuration cfg)
Carga la configuracion desde un archivo config.cfg.

Propiedades

- [VideoClickedEvent](#) **VideoClicked** [get]
Retorna el evento que se lanza cuando se hace click en el video.
- [VideoDraggedEvent](#) **VideoDragged** [get]
Retorna el evento que se lanza cuando se arrastra en el video.
- [VideoUppedEvent](#) **VideoUpped** [get]
Retorna el evento que se lanza cuando se deja de hacer click en el video.

Otros miembros heredados

3.22.1. Descripción detallada

Este script recoge eventos de raton sobre un video y los convierte en una serie de coordenadas normalizadas.

3.22.2. Documentación de las funciones miembro

3.22.2.1. LoadConfiguration()

```
override void InteractableOverlay.LoadConfiguration (
    Configuration cfg ) [virtual]
```

Carga la configuracion desde un archivo config.cfg.

Parámetros

<i>cfg</i>	El objeto Configuration incluye la configuracion de la aplicacion.
------------	--

Implementa [ConfigurableMonoBehaviour](#).

3.22.2.2. SetupCleanConfiguration()

```
override void InteractableOverlay.SetupCleanConfiguration (
    Configuration cfg ) [virtual]
```

Carga la configuracion por defecto en un objeto Configuration.

Parámetros

<i>cfg</i>	El objeto Configuration incluye la configuracion de la aplicacion.
------------	--

Implementa [ConfigurableMonoBehaviour](#).

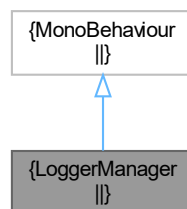
La documentación para esta clase fue generada a partir del siguiente fichero:

- Health-5G_Desktop/Assets/Scripts/Interaction/InteractableOverlay.cs

3.23. Referencia de la Clase LogManager

Apaga el log de Unity en la version compilada de la aplicacion.

Diagrama de herencias de LogManager



3.24 Referencia de la Clase MessageManager

29

3.23.1. Descripción detallada

Apaga el log de Unity en la version compilada de la aplicacion.

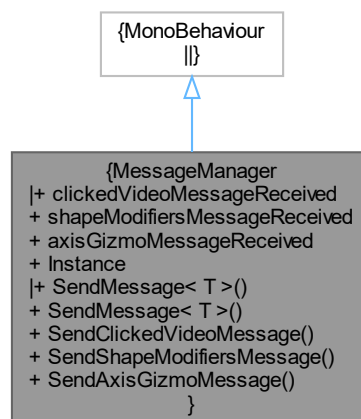
La documentación para esta clase fue generada a partir del siguiente fichero:

- Shared/Managers/LoggerManager.cs

3.24. Referencia de la Clase MessageManager

Facilita el envio y la recepcion de mensajes a traves de WebRTC.

Diagrama de herencias de MessageManager

**Métodos públicos**

- bool `SendMessage<T>(T msg, string channelLabel)`
Envia un mensaje.
- bool `SendMessage<T>(T msg, Microsoft.MixedReality.WebRTC.DataChannel channel)`
Envia un mensaje.
- bool `SendClickedVideoMessage(ClickedVideoMessage msg)`
Envia un mensaje.
- bool `SendShapeModifiersMessage(ShapeModifiersMessage msg)`
Envia un mensaje.
- bool `SendAxisGizmoMessage(AxisGizmoMessage msg)`
Envia un mensaje.

30

Atributos públicos

- **ClickedVideoMessageReceivedEvent clickedVideoMessageReceived** = new **ClickedVideoMessageReceivedEvent()**
El evento *ClickedVideoMessageReceivedEvent* se invoca cuando llega un mensaje de tipo *ClickedVideoMessage*.
- **ShapeModifiersMessageReceivedEvent shapeModifiersMessageReceived** = new **ShapeModifiersMessageReceivedEvent()**
El evento *ShapeModifiersMessageReceivedEvent* se invoca cuando llega un mensaje de tipo *ShapeModifiersMessage*.
- **AxisGizmoMessageReceivedEvent axisGizmoMessageReceived** = new **AxisGizmoMessageReceivedEvent()**
El evento *AxisGizmoMessageReceivedEvent* se invoca cuando llega un mensaje de tipo *AxisGizmoMessage*.

Propiedades

- static **MessageManager Instance** [get]
Retorna la instancia estática de este manager.

3.24.1. Descripción detallada

Facilita el envío y la recepción de mensajes a través de WebRTC.

3.24.2. Documentación de las funciones miembro

3.24.2.1. SendAxisGizmoMessage() bool MessageManager.SendAxisGizmoMessage (
 AxisGizmoMessage msg)

Envía un mensaje.

Parámetros

<i>msg</i>	El mensaje de tipo <i>AxisGizmoMessage</i> a enviar.
------------	--

Devuelve

True si se ha podido enviar, false si no.

3.24.2.2. SendClickedVideoMessage() bool MessageManager.SendClickedVideoMessage (
 ClickedVideoMessage msg)

Envía un mensaje.

Parámetros

<i>msg</i>	El mensaje de tipo <i>ClickedVideoMessage</i> a enviar.
------------	---

3.24 Referencia de la Clase MessageManager31

Devuelve
True si se ha podido enviar, false si no.

3.24.2.3. **SendMessage<T>()** [1/2] `bool MessageManager.SendMessage< T > (T msg, Microsoft.MixedReality.WebRTC.DataChannel channel)`

Envía un mensaje.

Parámetros del template

<i>T</i>	.
----------	---

Parámetros

<i>msg</i>	The msgT.
<i>channel</i>	El canal Microsoft.MixedReality.WebRTC.DataChannel por el que se envía el mensaje.

Devuelve
True si se ha podido enviar, false si no.

3.24.2.4. **SendMessage<T>()** [2/2] `bool MessageManager.SendMessage< T > (T msg, string channelLabel)`

Envía un mensaje.

Parámetros del template

<i>T</i>	.
----------	---

Parámetros

<i>msg</i>	El mensaje T a enviar.
<i>channelLabel</i>	Identificador del canal.

Devuelve
True si se ha podido enviar, false si no.

3.24.2.5. SendShapeModifiersMessage() `bool MessageManager.SendShapeModifiersMessage (ShapeModifiersMessage msg)`

Envia un mensaje.

Parámetros

<i>msg</i>	El mensaje de tipo ShapeModifiersMessage a enviar.
------------	--

Devuelve

True si se ha podido enviar, false si no.

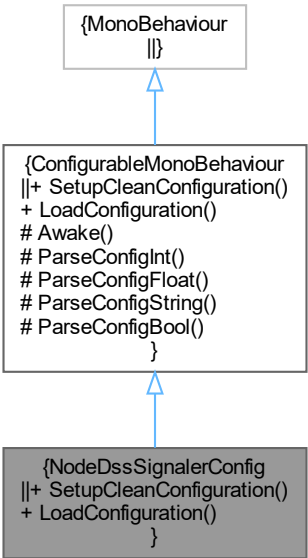
La documentación para esta clase fue generada a partir del siguiente fichero:

- Shared/Managers/MessageManager.cs

3.25. Referencia de la Clase NodeDssSignalerConfig

Configura un objeto de tipo NodeDssSignaler.

Diagrama de herencias de NodeDssSignalerConfig



3.25 Referencia de la Clase NodeDssSignalerConfig

33

Métodos públicos

- override void [SetupCleanConfiguration](#) (Configuration cfg)
Carga la configuracion por defecto en un objeto Configuration.
- override void [LoadConfiguration](#) (Configuration cfg)
Carga la configuracion desde un archivo config.cfg.

Otros miembros heredados

3.25.1. Descripción detallada

Configura un objeto de tipo NodeDssSignaler.

3.25.2. Documentación de las funciones miembro

3.25.2.1. LoadConfiguration() `override void NodeDssSignalerConfig.LoadConfiguration (Configuration cfg) [virtual]`

Carga la configuracion desde un archivo config.cfg.

Parámetros

<i>cfg</i>	El objeto Configuration incluye la configuracion de la aplicacion.
------------	--

Implementa [ConfigurableMonoBehaviour](#).

3.25.2.2. SetupCleanConfiguration() `override void NodeDssSignalerConfig.SetupCleanConfiguration (Configuration cfg) [virtual]`

Carga la configuracion por defecto en un objeto Configuration.

Parámetros

<i>cfg</i>	El objeto Configuration incluye la configuracion de la aplicacion.
------------	--

Implementa [ConfigurableMonoBehaviour](#).

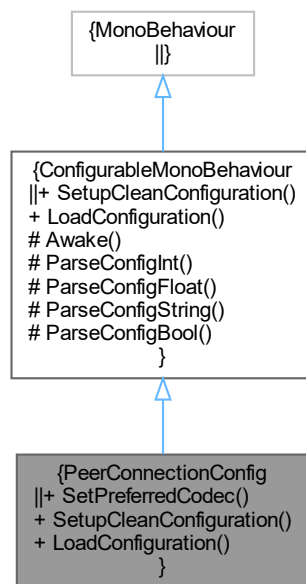
La documentación para esta clase fue generada a partir del siguiente fichero:

- Shared/Configuration/NodeDssSignalerConfig.cs

3.26. Referencia de la Clase PeerConnectionConfig

Configura un objeto de tipo PeerConnection.

Diagrama de herencias de PeerConnectionConfig



Métodos públicos

- void **SetPreferredCodec** ()
Cambia el codec preferido.
- override void **SetupCleanConfiguration** (Configuration cfg)
Carga la configuracion por defecto en un objeto Configuration.
- override void **LoadConfiguration** (Configuration cfg)
Carga la configuracion desde un archivo config.cfg.

Otros miembros heredados

3.26.1. Descripción detallada

Configura un objeto de tipo PeerConnection.

3.26.2. Documentación de las funciones miembro

3.26.2.1. LoadConfiguration() `override void PeerConnectionConfig.LoadConfiguration (Configuration cfg) [virtual]`

Carga la configuracion desde un archivo config.cfg.

Parámetros

cfg	El objeto Configuration incluye la configuracion de la aplicacion.
-----	--

Implementa [ConfigurableMonoBehaviour](#).

3.26.2.2. SetupCleanConfiguration() `override void PeerConnectionConfig.SetupCleanConfiguration (Configuration cfg) [virtual]`

Carga la configuracion por defecto en un objeto Configuration.

Parámetros

cfg	El objeto Configuration incluye la configuracion de la aplicacion.
-----	--

Implementa [ConfigurableMonoBehaviour](#).

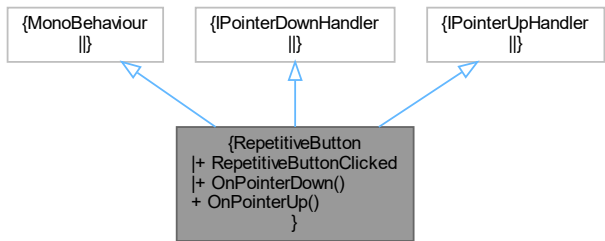
La documentación para esta clase fue generada a partir del siguiente fichero:

- Shared/Configuration/PeerConnectionConfig.cs

3.27. Referencia de la Clase RepetitiveButton

Script que permite que un boton actue repetidamente mientras se tiene pulsado.

Diagrama de herencias de RepetitiveButton



36

Métodos públicos

- void [OnPointerDown](#) (PointerEventData data)
El boton se ha pulsado.
- void [OnPointerUp](#) (PointerEventData data)
El boton se ha dejado de pulsar.

Propiedades

- [RepetitiveButtonClickedEvent](#) **RepetitiveButtonClicked** [get]
Evento que se lanzara mientras se tenga el boton pulsado.

3.27.1. Descripción detallada

Script que permite que un boton actue repetidamente mientras se tiene pulsado.

3.27.2. Documentación de las funciones miembro

3.27.2.1. OnPointerDown() void RepetitiveButton.OnPointerDown (
PointerEventData data)

El boton se ha pulsado.

Parámetros

<i>data</i>	Objeto PointerEventData del evento.
-------------	-------------------------------------

3.27.2.2. OnPointerUp() void RepetitiveButton.OnPointerUp (
PointerEventData data)

El boton se ha dejado de pulsar.

Parámetros

<i>data</i>	Objeto PointerEventData del evento.
-------------	-------------------------------------

La documentación para esta clase fue generada a partir del siguiente fichero:

- Shared/UserInterface/RepetitiveButton.cs

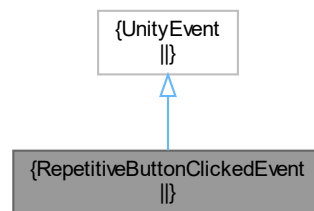
3.28 Referencia de la Clase RepetitiveButtonClickedEvent

37

3.28. Referencia de la Clase RepetitiveButtonClickedEvent

Evento que se lanzara mientras se tenga el boton pulsado.

Diagrama de herencias de RepetitiveButtonClickedEvent

**3.28.1. Descripción detallada**

Evento que se lanzara mientras se tenga el boton pulsado.

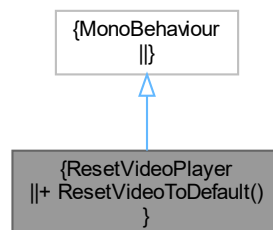
La documentación para esta clase fue generada a partir del siguiente fichero:

- Shared/UserInterface/RepetitiveButton.cs

3.29. Referencia de la Clase ResetVideoPlayer

Resetea varios Renderer con video YUV a un color por defecto.

Diagrama de herencias de ResetVideoPlayer



38

Métodos públicos

- void **ResetVideoToDefault** ()
Resetea los renderers a un color por defecto.

3.29.1. Descripción detallada

Resetea varios Renderer con video YUV a un color por defecto.

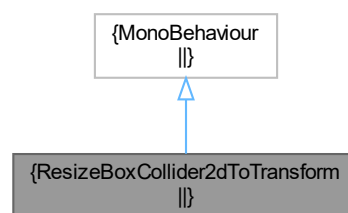
La documentación para esta clase fue generada a partir del siguiente fichero:

- Shared/UserInterface/ResetVideoPlayer.cs

3.30. Referencia de la Clase ResizeBoxCollider2dToTransform

Modifica un collider para que encaje con una transformacion.

Diagrama de herencias de ResizeBoxCollider2dToTransform



3.30.1. Descripción detallada

Modifica un collider para que encaje con una transformacion.

La documentación para esta clase fue generada a partir del siguiente fichero:

- Health-5G_Desktop/Assets/Scripts/Collider/ResizeBoxCollider2dToTransform.cs

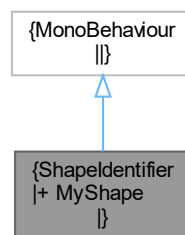
3.31 Referencia de la Clase Shapelfidentifier

39

3.31. Referencia de la Clase Shapelfidentifier

Identifica este marcador para saber si es una flecha, una caja, una cruz o un dibujo.

Diagrama de herencias de Shapelfidentifier

**Propiedades**

- Shape **MyShape** [get]
Retorna el identificador de la forma.

3.31.1. Descripción detallada

Identifica este marcador para saber si es una flecha, una caja, una cruz o un dibujo.

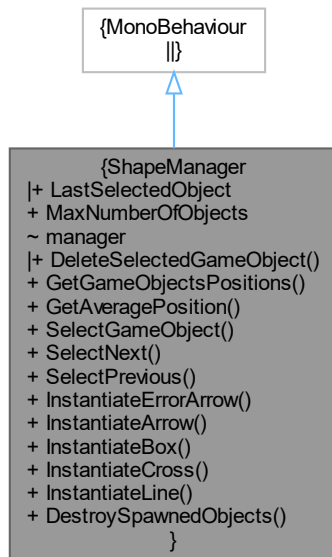
La documentación para esta clase fue generada a partir del siguiente fichero:

- Health-5G_Holo/Assets/Scripts/GameObjects/Shapelfidentifier.cs

3.32. Referencia de la Clase ShapeManager

Genera marcadores y ofrece utilidades para borrarlos y seleccionarlos.

Diagrama de herencias de ShapeManager



Métodos públicos

- void **DeleteSelectedGameObject** ()
Borra el objeto seleccionado.
- List< Vector3 > **GetGameObjectsPositions** (GameObject ignoreGameObject)
Retorna las posiciones de todos los objetos.
- bool **GetAveragePosition** (out Vector3 averagePoint, GameObject ignoreGameObject)
Calcula la posición media de todos los objetos.
- bool **SelectGameObject** (ShapelIdentifier thisIdentifier)
Selecciona un objeto en base a su ShapelIdentifier.
- void **SelectNext** ()
Selecciona el siguiente objeto.
- void **SelectPrevious** ()
Selecciona el objeto anterior.
- GameObject **InstantiateErrorArrow** ()
Instancia una flecha de error.
- GameObject **InstantiateArrow** ()
Instancia una flecha.
- GameObject **InstantiateBox** ()
Instancia una caja.
- GameObject **InstantiateCross** ()
Instancia una cruz.

3.32 Referencia de la Clase ShapeManager**41**

- GameObject **InstantiateLine** ()
Instancia una línea.
- void **DestroySpawnedObjects** ()
Borra todos los objetos de la escena.

Propiedades

- GameObject **LastSelectedObject** [get]
Devuelve el objeto seleccionado.
- int **MaxNumberOfObjects** [get, set]
Retorna el numero maximo de objetos en la escena.

3.32.1. Descripción detallada

Genera marcadores y ofrece utilidades para borrarlos y seleccionarlos.

3.32.2. Documentación de las funciones miembro

3.32.2.1. GetAveragePosition() `bool ShapeManager.GetAveragePosition (`
`out Vector3 averagePoint,`
`GameObject ignoreGameObject)`

Calcula la posicion media de todos los objetos.

Parámetros

<i>averagePoint</i>	Posicion media de los objetos.
<i>ignoreGameObject</i>	Objeto a ignorar.

Devuelve

True si se ha podido calcular una posicion media, false si no.

3.32.2.2. GetGameObjectsPositions() `List< Vector3 > ShapeManager.GetGameObjectsPositions (`
`GameObject ignoreGameObject)`

Retorna las posiciones de todos los objetos.

Parámetros

<i>ignoreGameObject</i>	Objeto a ignorar.
-------------------------	-------------------

42

Devuelve

Lista de posiciones de objetos.

3.32.2.3. InstantiateArrow() `GameObject ShapeManager.InstantiateArrow ()`

Instancia una flecha.

Devuelve

El GameObject instanciado.

3.32.2.4. InstantiateBox() `GameObject ShapeManager.InstantiateBox ()`

Instancia una caja.

Devuelve

El GameObject instanciado.

3.32.2.5. InstantiateCross() `GameObject ShapeManager.InstantiateCross ()`

Instancia una cruz.

Devuelve

El GameObject instanciado.

3.32.2.6. InstantiateErrorArrow() `GameObject ShapeManager.InstantiateErrorArrow ()`

Instancia una flecha de error.

Devuelve

El GameObject instanciado.

3.32.2.7. InstantiateLine() `GameObject ShapeManager.InstantiateLine ()`

Instancia una linea.

Devuelve

El GameObject instanciado.

3.32.2.8. SelectGameObject() `bool ShapeManager.SelectGameObject (
 ShapeIdentifier thisIdentifier)`

Selecciona un objeto en base a su [Shapeldentifier](#).

3.33 Referencia de la Clase ShapeModifiersMessage

43

Parámetros

<i>thisIdentifier</i>	El ShapeIdentifier del objeto.
-----------------------	--

Devuelve

True si se ha podido seleccionar, false si no.

La documentación para esta clase fue generada a partir del siguiente fichero:

- Health-5G_Holo/Assets/Scripts/Managers/ShapeManager.cs

3.33. Referencia de la Clase ShapeModifiersMessage

Mensaje que se envia al utilizar las herramientas de modificacion de marcadores.

Atributos públicos

- int **selectedTool**
Herramienta seleccionada Scale = 0, Position = 2, SelectNext = 3, SelectPrevious = 4, Reset = 5, Delete = 6
- int **selectedModifier**
Modificador More = 0, Less = 1
- bool **selectedXAxis**
Aplicar transformaciones en el eje X
- bool **selectedYAxis**
Aplicar transformaciones en el eje Y
- bool **selectedZAxis**
Aplicar transformaciones en el eje Z

3.33.1. Descripción detallada

Mensaje que se envia al utilizar las herramientas de modificacion de marcadores.

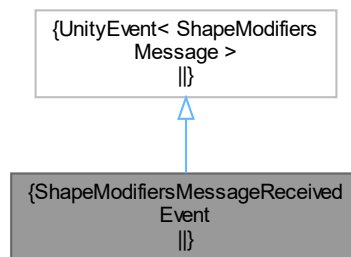
La documentación para esta clase fue generada a partir del siguiente fichero:

- Shared/Messages/ShapeModifiersMessage.cs

3.34. Referencia de la Clase ShapeModifiersMessageReceivedEvent

El evento [ShapeModifiersMessageReceivedEvent](#) se invoca cuando llega un mensaje de tipo [ShapeModifiersMessage](#).

Diagrama de herencias de ShapeModifiersMessageReceivedEvent



3.34.1. Descripción detallada

El evento [ShapeModifiersMessageReceivedEvent](#) se invoca cuando llega un mensaje de tipo [ShapeModifiersMessage](#).

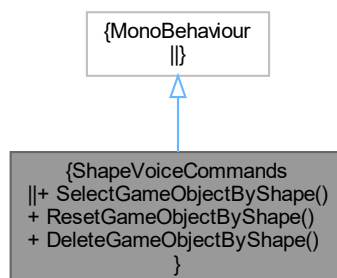
La documentación para esta clase fue generada a partir del siguiente fichero:

- Shared/Managers/MessageManager.cs

3.35. Referencia de la Clase ShapeVoiceCommands

Ofrece comandos de voz para marcadores 3D.

Diagrama de herencias de ShapeVoiceCommands



3.36 Referencia de la Clase SpaceObjectsManager

45

Métodos públicos

- void **SelectGameObjectByShape** ()
Selecciona este objeto.
- void **ResetGameObjectByShape** ()
Resetea este objeto.
- void **DeleteGameObjectByShape** ()
Borra este objeto.

3.35.1. Descripción detallada

Ofrece comandos de voz para marcadores 3D.

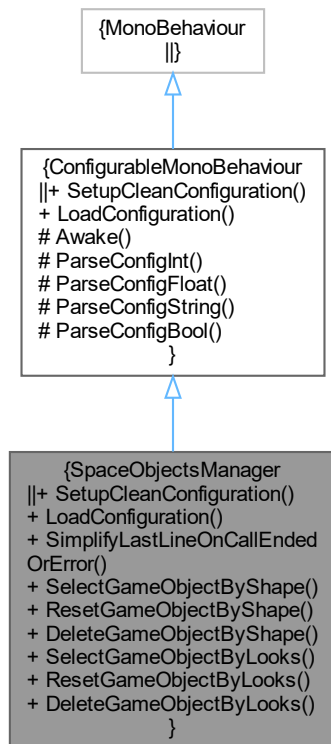
La documentación para esta clase fue generada a partir del siguiente fichero:

- Health-5G_Holo/Assets/Scripts/GameObjects/ShapeVoiceCommands.cs

3.36. Referencia de la Clase SpaceObjectsManager

Manager que se encarga del emplazamiento de los marcadores en el espacio.

Diagrama de herencias de SpaceObjectsManager

**Métodos públicos**

- override void [SetupCleanConfiguration](#) (Configuration cfg)
Carga la configuracion por defecto en un objeto Configuration.
- override void [LoadConfiguration](#) (Configuration cfg)
Carga la configuracion desde un archivo config.cfg.
- void **SimplifyLastLineOnCallEndedOrError** ()
Simplifica la ultima linea creada si se cae la llamada.
- void [SelectGameObjectByShape](#) (ShapeIdentifier sp)
Selecciona un objeto en base a su ShapeIdentifier.
- void [ResetGameObjectByShape](#) (ShapeIdentifier sp)
Resetea un objeto en base a su ShapeIdentifier.
- void [DeleteGameObjectByShape](#) (ShapeIdentifier sp)
Borra un objeto en base a su ShapeIdentifier.
- void **SelectGameObjectByLooks** ()

3.36 Referencia de la Clase SpaceObjectsManager

47

Selecciona un objeto en base a la direccion donde mira el usuario.

- void **ResetGameObjectByLooks** ()

Resetea un objeto en base a la direccion donde mira el usuario.

- void **DeleteGameObjectByLooks** ()

Borra un objeto en base a la direccion donde mira el usuario.

Otros miembros heredados

3.36.1. Descripción detallada

Manager que se encarga del emplazamiento de los marcadores en el espacio.

3.36.2. Documentación de las funciones miembro

3.36.2.1. DeleteGameObjectByShape() void SpaceObjectsManager.DeleteGameObjectByShape (
 [ShapeIdentifier](#) sp)

Borra un objeto en base a su [ShapeIdentifier](#).

Parámetros

<i>sp</i>	El ShapeIdentifier del objeto.
-----------	--

3.36.2.2. LoadConfiguration() override void SpaceObjectsManager.LoadConfiguration (
 Configuration *cfg*) [virtual]

Carga la configuracion desde un archivo config.cfg.

Parámetros

<i>cfg</i>	El objeto Configuration incluye la configuracion de la aplicacion.
------------	--

Implementa [ConfigurableMonoBehaviour](#).

3.36.2.3. ResetGameObjectByShape() void SpaceObjectsManager.ResetGameObjectByShape (
 [ShapeIdentifier](#) sp)

Resetea un objeto en base a su [ShapeIdentifier](#).

48

Parámetros

<i>sp</i>	El ShapeIdentifier del objeto.
-----------	--

3.36.2.4. SelectGameObjectByShape() `void SpaceObjectsManager.SelectGameObjectByShape (
 ShapeIdentifier sp)`

Selecciona un objeto en base a su [ShapeIdentifier](#).

Parámetros

<i>sp</i>	El ShapeIdentifier del objeto.
-----------	--

3.36.2.5. SetupCleanConfiguration() `override void SpaceObjectsManager.SetupCleanConfiguration (
 Configuration cfg) [virtual]`

Carga la configuracion por defecto en un objeto Configuration.

Parámetros

<i>cfg</i>	El objeto Configuration incluye la configuracion de la aplicacion.
------------	--

Implementa [ConfigurableMonoBehaviour](#).

La documentación para esta clase fue generada a partir del siguiente fichero:

- Health-5G_Holo/Assets/Scripts/Managers/SpaceObjectsManager.cs

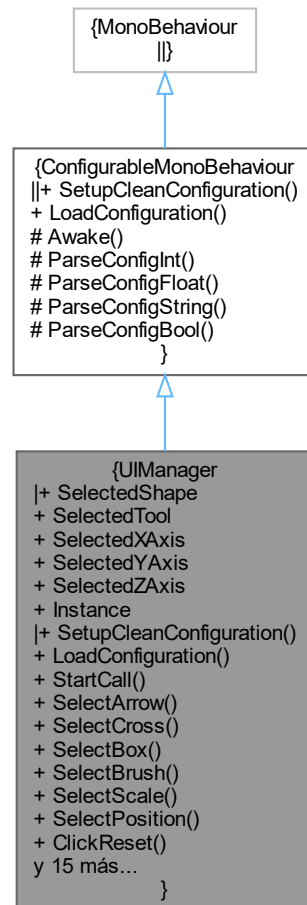
3.37. Referencia de la Clase UIManager

Manager de la UI de la app de escritorio.

3.37 Referencia de la Clase UIManager

49

Diagrama de herencias de UIManager



Métodos públicos

- override void **SetupCleanConfiguration** (Configuration cfg)
Carga la configuracion por defecto en un objeto Configuration.
- override void **LoadConfiguration** (Configuration cfg)
Carga la configuracion desde un archivo config.cfg.
- void **StartCall** ()
Inicia una llamada.
- void **SelectArrow** ()
Selecciona el marcador de tipo flecha.

- void **SelectCross** ()
Selecciona el marcador de tipo cruz.
- void **SelectBox** ()
Selecciona el marcador de tipo caja.
- void **SelectBrush** ()
Selecciona el marcador de tipo pincel.
- void **SelectScale** ()
Selecciona la herramienta de escala.
- void **SelectPosition** ()
Selecciona la herramienta de posicion.
- void **ClickReset** ()
Selecciona la herramienta de reset.
- void **ClickDelete** ()
Selecciona la herramienta de borrado.
- void **ClickXaxis** ()
Selecciona la herramienta de eje X.
- void **ClickYaxis** ()
Selecciona la herramienta de eje Y.
- void **ClickZaxis** ()
Selecciona la herramienta de eje Z.
- void **ClickLess** ()
Aplica la herramienta de forma negativa.
- void **ClickMore** ()
Aplica la herramienta de forma positiva.
- void **ClickSelectPrevious** ()
Selecciona el marcador anterior.
- void **ClickSelectNext** ()
Selecciona el marcador siguiente.
- void **OnCall** ()
Se ejecuta al iniciar una llamada.
- void **Calling** ()
Se ejecuta mientras se inicia una llamada.
- void **EndCall** ()
Se ejecuta al acabar una llamada.
- void **OnVideoClicked** (float x, float y)
Se llama cuando el usuario hace click sobre el video.
- void **OnVideoDragged** (float x, float y)
Se llama cuando el usuario hace drag sobre el video.
- void **OnVideoUpped** (float x, float y)
Se llama cuando el usuario deja de hacer click sobre el video.
- void **ResetAxis** ()
Retea los botones de seleccion de eje.

Propiedades

- Shape **SelectedShape** [get]
Retorna la forma seleccionada.
- Tool **SelectedTool** [get]
Retorna la herramienta seleccionada.
- bool **SelectedXAxis** [get]

3.37 Referencia de la Clase UIManager**51**

Retorna si el eje X esta seleccionado o no.

- bool **SelectedYAxis** [get]

Retorna si el eje Y esta seleccionado o no.

- bool **SelectedZAxis** [get]

Retorna si el eje Z esta seleccionado o no.

- static **UIManager Instance** [get]

Retorna la instancia de este manager.

Otros miembros heredados**3.37.1. Descripción detallada**

Manager de la UI de la app de escritorio.

3.37.2. Documentación de las funciones miembro

3.37.2.1. LoadConfiguration() `override void UIManager.LoadConfiguration (Configuration cfg) [virtual]`

Carga la configuracion desde un archivo config.cfg.

Parámetros

<i>cfg</i>	El objeto Configuration incluye la configuracion de la aplicacion.
------------	--

Implementa [ConfigurableMonoBehaviour](#).

3.37.2.2. OnVideoClicked() `void UIManager.OnVideoClicked (float x, float y)`

Se llama cuando el usuario hace click sobre el video.

Parámetros

<i>x</i>	Coordenada X de la posicion.
<i>y</i>	Coordenada y de la posicion.

3.37.2.3. OnVideoDragged() `void UIManager.OnVideoDragged (float x, float y)`

52

Se llama cuando el usuario hace drag sobre el video.

Parámetros

<i>x</i>	Coordenada X de la posicion.
<i>y</i>	Coordenada y de la posicion.

3.37.2.4. OnVideoUpped() `void UIManager.OnVideoUpped (`
 `float x,`
 `float y)`

Se llama cuando el usuario deja de hacer click sobre el video.

Parámetros

<i>x</i>	Coordenada X de la posicion.
<i>y</i>	Coordenada y de la posicion.

3.37.2.5. SetupCleanConfiguration() `override void UIManager.SetupCleanConfiguration (`
 `Configuration cfg) [virtual]`

Carga la configuracion por defecto en un objeto Configuration.

Parámetros

<i>cfg</i>	El objeto Configuration incluye la configuracion de la aplicacion.
------------	--

Implementa [ConfigurableMonoBehaviour](#).

La documentación para esta clase fue generada a partir del siguiente fichero:

- Health-5G_Desktop/Assets/Scripts/Managers/UIManager.cs

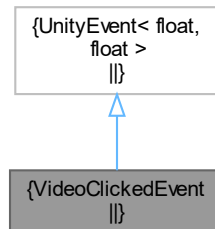
3.38. Referencia de la Clase VideoClickedEvent

Evento que se lanza cuando se hace click en el video.

3.39 Referencia de la Clase VideoDraggedEvent

53

Diagrama de herencias de VideoClickedEvent

**3.38.1. Descripción detallada**

Evento que se lanza cuando se hace click en el video.

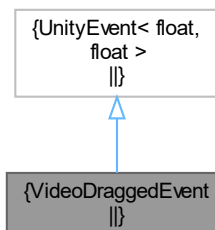
La documentación para esta clase fue generada a partir del siguiente fichero:

- Health-5G_Desktop/Assets/Scripts/Interaction/InteractableOverlay.cs

3.39. Referencia de la Clase VideoDraggedEvent

Evento que se lanza cuando se arrastra en el video.

Diagrama de herencias de VideoDraggedEvent



54

3.39.1. Descripción detallada

Evento que se lanza cuando se arrastra en el video.

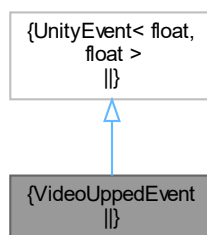
La documentación para esta clase fue generada a partir del siguiente fichero:

- Health-5G_Desktop/Assets/Scripts/Interaction/InteractableOverlay.cs

3.40. Referencia de la Clase VideoUppedEvent

Evento que se lanza cuando se deja de hacer click en el video.

Diagrama de herencias de VideoUppedEvent



3.40.1. Descripción detallada

Evento que se lanza cuando se deja de hacer click en el video.

La documentación para esta clase fue generada a partir del siguiente fichero:

- Health-5G_Desktop/Assets/Scripts/Interaction/InteractableOverlay.cs

Índice alfabético

ArrowAnimation, [5](#)
 AxisGizmoManager, [6](#)
 OnMessageReceived, [6](#)
 AxisGizmoMessage, [7](#)
 AxisGizmoMessageReceivedEvent, [7](#)

 CallButtonClickedEvent, [8](#)
 CallEndedEvent, [8](#)
 CallFailedEvent, [9](#)
 CallStartedEvent, [9](#)
 CleanUpManager, [10](#)
 ClickedVideoMessage, [11](#)
 ClickedVideoMessageReceivedEvent, [11](#)
 ConfigurableMonoBehaviour, [12](#)
 LoadConfiguration, [13](#)
 ParseConfigBool, [13](#)
 ParseConfigFloat, [13](#)
 ParseConfigInt, [13](#)
 ParseConfigString, [14](#)
 SetupCleanConfiguration, [14](#)
 ConnectionManager, [14](#)
 GetOrCreateDataChannel, [16](#)
 GetOrCreateDataChannelUnreliable, [16](#)
 SendMessage< T >, [17](#)

 DataChannelAddedEvent, [18](#)
 DeleteGameObjectByShape
 SpaceObjectsManager, [47](#)
 DemoSignalingServer, [18](#)

 FacePlayerAnimation, [20](#)

 GetAveragePosition
 ShapeManager, [41](#)
 GetGameObjectsPositions
 ShapeManager, [41](#)
 GetOrCreateDataChannel
 ConnectionManager, [16](#)
 GetOrCreateDataChannelUnreliable
 ConnectionManager, [16](#)

 HandMenuFunctions, [20](#)

 ImageDownloaderManager, [22](#)
 ImageDownloaderManagerCanvas, [23](#)
 ImageDownloadManagerHub, [24](#)
 LoadConfiguration, [25](#)
 SetupCleanConfiguration, [25](#)
 InitialPositionAndScale, [25](#)
 InstantiateArrow
 ShapeManager, [42](#)
 InstantiateBox
 ShapeManager, [42](#)
 InstantiateCross
 ShapeManager, [42](#)
 InstantiateErrorArrow
 ShapeManager, [42](#)
 InstantiateLine
 ShapeManager, [42](#)
 InteractableOverlay, [26](#)
 LoadConfiguration, [28](#)
 SetupCleanConfiguration, [28](#)

 LoadConfiguration
 ConfigurableMonoBehaviour, [13](#)
 ImageDownloadManagerHub, [25](#)
 InteractableOverlay, [28](#)
 NodeDssSignalerConfig, [33](#)
 PeerConnectionConfig, [35](#)
 SpaceObjectsManager, [47](#)
 UIManager, [51](#)
 LoggerManager, [28](#)

 MessageManager, [29](#)
 SendAxisGizmoMessage, [30](#)
 SendClickedVideoMessage, [30](#)
 SendMessage< T >, [31](#)
 SendShapeModifiersMessage, [31](#)

 NodeDssSignalerConfig, [32](#)
 LoadConfiguration, [33](#)
 SetupCleanConfiguration, [33](#)

 OnMessageReceived
 AxisGizmoManager, [6](#)
 OnPointerDown
 RepetitiveButton, [36](#)
 OnPointerUp
 RepetitiveButton, [36](#)
 OnVideoClicked
 UIManager, [51](#)
 OnVideoDragged
 UIManager, [51](#)
 OnVideoUpped
 UIManager, [52](#)

 ParseConfigBool
 ConfigurableMonoBehaviour, [13](#)
 ParseConfigFloat
 ConfigurableMonoBehaviour, [13](#)
 ParseConfigInt
 ConfigurableMonoBehaviour, [13](#)
 ParseConfigString
 ConfigurableMonoBehaviour, [14](#)
 PeerConnectionConfig, [34](#)
 LoadConfiguration, [35](#)
 SetupCleanConfiguration, [35](#)

 RepetitiveButton, [35](#)
 OnPointerDown, [36](#)
 OnPointerUp, [36](#)
 RepetitiveButtonClickedEvent, [37](#)

- ResetGameObjectByShape
 - SpaceObjectsManager, [47](#)
- ResetVideoPlayer, [37](#)
- ResizeBoxCollider2dToTransform, [38](#)
- SelectGameObject
 - ShapeManager, [42](#)
- SelectGameObjectByShape
 - SpaceObjectsManager, [48](#)
- SendAxisGizmoMessage
 - MessageManager, [30](#)
- SendClickedVideoMessage
 - MessageManager, [30](#)
- SendMessage< T >
 - ConnectionManager, [17](#)
 - MessageManager, [31](#)
- SendShapeModifiersMessage
 - MessageManager, [31](#)
- SetupCleanConfiguration
 - ConfigurableMonoBehaviour, [14](#)
 - ImageDownloadManagerHub, [25](#)
 - InteractiveOverlay, [28](#)
 - NodeDssSignalerConfig, [33](#)
 - PeerConnectionConfig, [35](#)
 - SpaceObjectsManager, [48](#)
 - UIManager, [52](#)
- ShapeIdentifier, [39](#)
- ShapeManager, [39](#)
 - GetAveragePosition, [41](#)
 - GetGameObjectsPositions, [41](#)
 - InstantiateArrow, [42](#)
 - InstantiateBox, [42](#)
 - InstantiateCross, [42](#)
 - InstantiateErrorArrow, [42](#)
 - InstantiateLine, [42](#)
 - SelectGameObject, [42](#)
- ShapeModifiersMessage, [43](#)
- ShapeModifiersMessageReceivedEvent, [44](#)
- ShapeVoiceCommands, [44](#)
- SpaceObjectsManager, [45](#)
 - DeleteGameObjectByShape, [47](#)
 - LoadConfiguration, [47](#)
 - ResetGameObjectByShape, [47](#)
 - SelectGameObjectByShape, [48](#)
 - SetupCleanConfiguration, [48](#)
- UIManager, [48](#)
 - LoadConfiguration, [51](#)
 - OnVideoClicked, [51](#)
 - OnVideoDragged, [51](#)
 - OnVideoUpped, [52](#)
 - SetupCleanConfiguration, [52](#)
- VideoClickedEvent, [52](#)
- VideoDraggedEvent, [53](#)
- VideoUppedEvent, [54](#)