UNIVERSIDAD DE CASTILLA-LA MANCHA

ESCUELA SUPERIOR DE INFORMÁTICA

GRADO EN INGENIERÍA INFORMÁTICA

Tecnología Específica de Ingeniería del Software

TRABAJO FIN DE GRADO

Plataforma para la convergencia tecnológica en el desarrollo de aplicaciones industriales de realidad mixta (Apholo)

Rubén Moraleda Sánchez

julio, 2022

UNIVERSIDAD DE CASTILLA-LA MANCHA
ESCUELA SUPERIOR DE INFORMÁTICA

Departamento de Tecnologías y Sistemas de la Información

Tecnología específica de Ingeniería del Software

TRABAJO FIN DE GRADO

Plataforma para la convergencia tecnológica en el desarrollo de aplicaciones industriales de realidad mixta (Apholo)

Autor: Rubén Moraleda Sánchez

Tutor(a): Dr. Carlos González Morcillo

Co-tutor(a): Francisco Manuel García Sánchez-Belmonte

julio, 2022

Plataforma para la convergencia tecnológica en el desarrollo aplicaciones industriales de realidad mixta (Apholo)
© Rubén Moraleda Sánchez, 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/.

El escudo basado en el núcleo de ferrita que acompaña la distribución de este documento ha sido realizado por Francisco Moya, David Villa e Ignacio Díez y su inclusión en el documento opcional debe respetar los derechos de la licencia CC BY-SA 3.0 con la que se distribuye

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 de TFG para la UCLM publicada por Jesús Salido en GitHub[1] y Overleaf [2] 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.

[1] https://github.com/JesusSalido/TFG_ESI_UCLM, DOI: 10.5281/zenodo.4574562
[2] https://www.overleaf.com/latex/templates/plantilla-de-tfg-escuela-superior-de-informatica-uclm/phjgscmfqtsw

Tribunal:

Presidente: _____

Vocal: _____

Secretario(a): _____

Fecha de defensa: _____

Calificación: _____

Presidente	Vocal	Secretario(a)
Fdo.:	Fdo.:	Fdo.:

TRABAJO FIN DE GRADO - ESCUELA SUP. DE INFORMÁTICA (UCLM)

Plataforma para la convergencia tecnológica en el desarrollo aplicaciones industriales de realidad mixta (Apholo)

Rubén Moraleda Sánchez
Ciudad Real, julio 2022

Resumen

La realidad mixta surge como punto de inflexión entre el desarrollo de aplicaciones de Realidad Aumentada y Realidad Virtual. En la realidad mixta, los objetos, el entorno y los datos procedentes de sensores heterogéneos se superponen a la imagen del mundo real con la capacidad de interactuar entre sí.

En el ámbito industrial, la realidad mixta como innovación técnica ha permitido el desarrollo de novedosas aplicaciones. Sin embargo, los índices de implantación y productividad son mucho más modestos que en otros campos en los que la realidad mixta se ha ido implementando a gran velocidad (entretenimiento, medicina, educación). Es por ello que el margen de mejora para este sector es mucho mayor y las posibles innovaciones tendrán un impacto muy elevado.

Apholo surge como una solución al problema del importante esfuerzo que requiere el diseño, desarrollo y mantenimiento de aplicaciones de realidad mixta para la adaptación de aplicaciones ya existentes del sector industrial.

Platform for technological convergence in the development of mixed reality industrial applications (Apholo)

Rubén Moraleda Sánchez
Ciudad Real, July 2022

Abstract

Mixed reality emerges as a turning point between the development of Augmented Reality and Virtual Reality applications. In mixed reality, heterogeneous objects, environment, and sensor data are overlaid on the real-world image with the ability to interact with each other.

In the industrial field, mixed reality as a technical innovation has allowed the development of novel applications. However, the rates of implementation and productivity are much more modest than in other fields where mixed reality has been implemented at great speed (entertainment, medicine, education). That is why the room for improvement in this sector is much greater and possible innovations will have a very high impact.

Apholo arises as a solution to the problem of the important effort required by the design, development and maintenance of mixed reality applications for the adaptation of already existing applications in the industrial sector.

Agradecimientos

A mis padres, Jose María y Raquel, por haber sido un ejemplo para mí, por su apoyo incondicional y por su paciencia desmedida. Sin ellos nada hubiera sido posible.

A mis compañeros de la carrera, especialmente Pablo y Tomás, porque si no hubieran estado ahí hubiera sido prácticamente imposible avanzar curso a curso con determinación. Además, los momentos difíciles unen como ninguna otra cosa a las personas.

A mis profesores, porque han sabido transmitir sus conocimientos y su pasión por la ingeniería informática, habiendo quedado, espero, algo de poso en mi.

También quiero dar las gracias a mis amigos de fuera de la carrera, por haberme apoyado en los buenos momentos y en los no tan buenos, y porque si todavía seguimos juntos después de que cada uno haya elegido un camino diferente, es que son para siempre.

Y por último y no por ello menos importante, a Carlos y Paco, por haber confiado en mí, dirigiendo este proyecto y brindándome su confianza.

Rubén Moraleda Sánchez
Ciudad Real, 2022

Índice general

Índice de figuras

Índice de tablas

Índice de listados

CAPÍTULO 1
Introducción

La *realidad mixta* surge como punto de inflexión entre el desarrollo de aplicaciones de *realidad aumentada* y *realidad virtual*. En la realidad mixta, los objetos, el entorno y los datos procedentes de sensores heterogéneos se superponen a la imagen del mundo real con la capacidad de interactuar entre sí.

Ha sido fundamentalmente en el siglo XXI cuando estas tencologías están empezando a ser verdaderamente útiles. Los avances en la potencia de los procesadores, unidos al rápido aumento en la capacidad de almacenamiento, las mejoras en las conexiones inalámbricas con el 4G y 5G, la utilización masiva de los teléfonos móviles y el progreso de los servicios en la nube, ha permitido formar el caldo de cultivo necesario para empezar a desplegar todo el potencial de la realidad mixta.

En los últimos años, la realidad aumentada ha sido ampliamente implementada en numerosos campos, incluyendo el sector educativo, industrial, la construcción, la medicina y los videojuegos, mientras que la realidad virtual ha sido principalmente empleada en el sector del entretenimiento[27]. Así pues, comparada con la realidad aumentada, el número de campos en los que ha adoptado la realidad virtual es mucho menor. Por ello en parte, la diferenciación entre realidad aumentada y realidad mixta a veces es un poco ambigua, ya que la mayor parte de la realidad mixta se basa en realidad aumentada.

Los inicios del desarrollo de la realidad mixta podemos ubicarlos en la década de los 60 del siglo XX, con los primeros prototipos de cascos que utilizan esta tecnología [7]. Ya en los años 90 se observa un uso más extendido, especialmente en el campo aeroespacial.

Desde entonces se ha producido un incremento exponencial en el número de instituciones científicas, universidades y compañías que han invertido recursos en la investigación en realidad mixta. Destaca el lanzamiento en 2016 por parte de Microsoft de su primera versión de las gafas de realidad mixta *HoloLens*, que alcanzó unas cotas de popularidad nunca antes vistas. La realidad mixta dejaba ya de ser algo propio de la ciencia ficción para empezar a sentirse el aterrizaje de sus primeras aplicaciones prácticas.

En el ámbito industrial, la realidad mixta como innovación técnica ha permitido el desarrollo de novedosas aplicaciones[3]. Sin embargo, los índices de implantación y productividad son mucho más modestos que en otros campos en los que la realidad mixta se ha ido implementando a mayor velocidad (entretenimiento, medicina, educación). Es por ello que el margen de mejora para este sector es mucho mayor y las posibles innovaciones tendrán un impacto muy elevado.

1.1. CONTEXTO

Este proyecto se ha desarrollado en el marco de un *Trabajo Fin de Grado* en *Ingeniería informática* asociado a la realización de unas prácticas externas facilitadas por la *Escuela Superior de Informática* de Ciudad Real (Universidad de Castilla-La Mancha).

1.2. CONVENIO DE CONFIDENCIALIDAD

Este TFG está sujeto a un acuerdo de confidencialidad contraído con la empresa en la que se ha enmarcado su realización, Furious Koalas S.L.

Esto tiene como consecuencia la no inclusión en esta memoria de ninguna referencia gráfica o del código fuente completo de las aplicaciones desarrolladas. No obstante, este acuerdo si permite la entrega de ciertos anexos en formato físico el día de la defensa. Los anexos que se entregarán en formato físico son:

- Manual de usuario.
- Capturas de las aplicación y elementos gráficos que la componen.
- Documentación del código fuente.

Para más detalles se pueden revisar las clausulas completas en el anexo V entregado junto a esta memoria.

1.3. ENTORNO

El entorno de ejecución se compone de los siguientes dispositivos:

- Un ordenador, que se utilizará como anfitrión de la parte del sistema encargado de realizar las mayores operaciones de cálculo y procesamientos. Deberá de disponer de *webcam* para utilizar el módulo 2.
- Una unidad de *HoloLens 2*, gafas de realidad mixta que actualmente constituyen el estándar de facto en el sector, y que proveen la interfaz para la interacción del usuario con las aplicaciones desarrolladas.

1.4. PROBLEMÁTICA

Apholo surge como una solución al problema del importante esfuerzo que requiere el diseño, desarrollo y mantenimiento de aplicaciones de realidad mixta para la adaptación de aplicaciones ya existentes del sector industrial. Además, se pretende que el uso de *Apholo* mejore el desempeño de los trabajadores, pues el empleo de ordenadores convencionales en entornos industriales supone en numerosas ocasiones un lastre al rendimiento y la productividad, por razones fundamentalmente de usabilidad (necesidad de desplazarse a la ubicación del ordenador, tener las manos manchadas por el trabajo manual, etc.).

La solución a desarrollar se llevará a cabo mediante una división de la plataforma en tres módulos, cada uno con su prototipo de aplicación correspondiente:

- **Módulo 1: Aplicación interactiva mediante *streaming* de captura de pantalla**. Se tratará de una aplicación genérica para el *streaming* de la ventana de otra aplicación de escritorio existente del sector industrial, de tal forma que pueda ser utilizada de forma remota en el dispositivo de realidad mixta. Este módulo estará formado por:
    - Una **aplicación de escritorio** que será ejecutada en el ordenador anfitrión donde se utilice la aplicación existente que se pretende manejar de forma remota, encargada de recibir el flujo de interacción del usuario y transformarla en eventos de teclado y ratón. Además, también deberá ser capaz de capturar la ventana de la aplicación existente para enviarla y así poder ser mostrada en el dispositivo de realidad mixta.

    - Una **aplicación para *HoloLens 2*** que muestre la ventana de la aplicación existente en formato de holograma y permita la interacción del usuario mediante gestos con la mano y comandos de voz, los cuales serán enviados a la aplicación de escritorio para su conversión en eventos de teclado y ratón.

- **Módulo 2: Aplicación de soporte remoto por videoconferencia**. Este módulo estará compuesto por una aplicación que permita brindar soporte remoto por parte de un usuario de la aplicación de ordenador a otro usuario de la aplicación de *HoloLens 2*. El canal de la ayuda será a través de un holograma que muestre la imagen del técnico de soporte en el caso de la aplicación de *HoloLens 2*, y de una ventana con la imagen recogida por la cámara de las gafas en el caso de la aplicación de escritorio. Por lo tanto, la estructura sería similar al módulo anterior:
    - Una **aplicación de escritorio** que será ejecutada en el ordenador utilizado por el técnico de soporte, que recogerá el vídeo (imagen y sonido) mediante una *webcam* y lo transmitirá a las *HoloLens 2* y a su vez muestre el vídeo recogido por la aplicación de *HoloLens 2*.
    - Una **aplicación para *HoloLens 2*** que muestre la imagen y reproduzca el audio recibido de la aplicación de escritorio y a su vez recoja la imagen y audio mediante la cámara y micrófonos de las gafas y lo envíe a la aplicación de escritorio.
- **Módulo 3: Aplicación con componentes nativos de *HoloLens 2***. Se tratará de una aplicación construida mediante componentes nativos de *HoloLens 2*. Está aplicación estará basada en una aplicación existente del ámbito industrial, de tal modo que supondrá una conversión o versión de dicha aplicación para *HoloLens 2*, empleando así las posibilidades que ofrece la realidad mixta para mejorar el uso de esta aplicación en el entorno industrial. Este módulo estará formado por:
    - Una **aplicación de escritorio** encargada gestionar las peticiones al servidor web de la aplicación existente de ámbito industrial para obtener la información requerida por la aplicación de *HoloLens 2*. También realizará las tareas que requieran un mayor cómputo.
    - Una **aplicación para *HoloLens 2*** basada en una aplicación existente del sector industrial construida con componentes nativos de la biblioteca de realidad mixta MRTK de Microsoft. Esta aplicación deberá iniciar el flujo de peticiones al servidor web para obtener la información (texto, imágenes, etc.) que deberá mostrar al trabajador para el desempeño de sus tareas.

1.5. ESTRUCTURA DEL DOCUMENTO

El presente documento se ha organizado atendiendo a las indicaciones de la normativa de trabajos de fin de grado de la Escuela Superior de Informática de la Universidad de Castilla-La Mancha, empleando los siguientes capítulos.

Capítulo 2: Estado del Arte

En este capítulo se hace un repaso a los conocimientos antecedentes dentro del campo de la realidad mixta, primero de forma general y luego centrándolos en el sector industrial.

Capítulo 3: Objetivos

En este capítulo se enumeran y se describen los diferentes objetivos y subobjetivos plantea-dos para este TFG.

Capítulo 4: Metodología

En este capítulo se expone y se justifica la metodología elegida para el desarrollo de este sistema. También se describen los recursos empleados, tanto hardware como software.

Capítulo 5: Resultados

En este capítulo se enumeran y explican los resultados más destacables que se han obtenido en la elaboración del presente TFG.

Capítulo 6: Conclusiones

Finalmente se expone un resumen revisando los objetivos planteados a modo de conclusión, información sobre los costes y horas dedicadas, estadísticas sobre el repositorio y el código fuente, además de una serie de líneas de trabajo futuro planteadas para continuar su desarrollo y finalmente, una conclusión personal.

CAPÍTULO 2

Estado del arte

2.1. TERMINOLOGÍA: REALIDAD MIXTA, REALIDAD VIRTUAL, REALIDAD AUMENTADA

Antes de exponer la temática del presente epígrafe es conveniente realizar una breve aclaración sobre estos tres términos que a menudo son confundidos y utilizados indistintamente. Posteriormente se expondrán los antecedentes de estas tecnologías para conocer suficientemente el contexto del tema del presente trabajo.

Así pues, podemos empezar el abordaje de esta problemática afirmando que la *realidad mixta* constituye la suma de las características propias de la *realidad virtual* y la *realidad aumentada*. De aquí podemos inferir que se trata de la tecnología más moderna y compleja, pero es precio conocer en que consisten estas dos técnicas en las que se basa para saber a qué nos estamos refiriendo.

No obstante, hay otro término que engloba a todos los anteriores, y es el de *Realidad Extendida* o XR (*Extended Reality*), donde la «X» representa cualquier tecnología de computación espacial, presente o futura, tales como la realidad aumentada, la realidad mixta y la realidad virtual, y todas las tecnologías que se interpolan entre ellas [9].

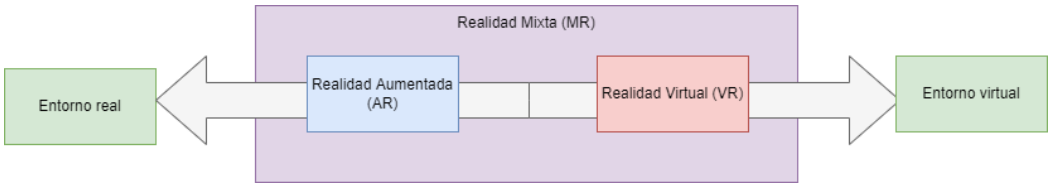

Figura 2.1: Diagrama de tecnologías de la Realidad Extendida

Ahora sí, se procede a la definición de cada una de estas tecnologías.

2.1.1. Realidad virtual

Comenzando por la realidad virtual, el objetivo de la misma es la inmersión total del usuario en un espacio radicalmente distinto del real. Para ello sustituye el entorno del individuo por una escena generada por ordenador o una localización real previamente capturada. Un ejemplo de este tipo de tecnología lo constituye el dispositivo *Oculus* de Facebook.

Esto se consigue a través de un visor de realidad virtual que está compuesto por auriculares y una pantalla que ocupa todo el rango de visión del usuario, canales a través de los cuales se consigue el bloqueo de los estímulos auditivos y visuales del mundo real, sustituyéndolos por los de la escena virtual que se desea generar. Para aumentar la experiencia inmersiva algunos dispositivos también incluyen controladores manuales para interactuar con el mundo artificial.

2.1.2. Realidad aumentada

Por su parte, la realidad aumentada no oculta ni remplaza el entorno del usuario, sino que superpone una capa de información visual y sonora sobre el entorno en tiempo real. De este modo la diferencia principal respecto a la realidad virtual es que en la realidad aumentada el usuario continúa percibiendo el mundo real, añadiendo elementos digitales que "aumentan" la información obtenida de un mismo entorno.

Un ejemplo de uso de esta tecnología serían las aplicaciones para móviles como *Pokemon Go* o la aplicación de transporte de Madrid que se ayuda de la realidad aumentada para mostrar información relativa a los horarios o paradas del colectivo.

2.1.3. Realidad mixta

Una vez conocidas las tecnologías en las que se basa podemos definir mejor en que consiste la realidad mixta.

La realidad mixta es la combinación de las mejores capacidades de la realidad virtual y la realidad aumentada. En consecuencia, permite al usuario percibir el mundo real al tiempo que visualiza una capa de información extra, pero además los objetos virtuales que se renderizan se integran perfectamente con el mundo real. Esto significa que, si en nuestra escena tenemos una pelota 3D debajo de una mesa, la pelota quedaría oculta por la mesa a no ser que nos agacháramos para observarla. También los objetos virtuales son afectados por la profundidad y la perspectiva, renderizándose ante el usuario en función de la posición que ocupe.

Cabe destacar que existe la opinión de que la creación del término «realidad mixta» obedece más a cuestiones de *marketing* que a diferencias notables con la realidad aumentada que justifiquen una nueva clasificación[5].

Sin embargo, y aunque esto pueda ser cierto, el concepto de realidad mixta es ampliamente aceptado y utilizado por una gran variedad de empresas tecnológicas, como Meta y su *Oculus Quest 2*, Microsoft y sus *Hololens 2* y *Magic Leap* y su dispositivo homónimo.

2.2. ANTECEDENTES

2.2.1. Antecedentes generales

Pasando a los antecedentes, podemos remontarnos tan atrás en el tiempo como el siglo III a.C., cuando Euclides da los primeros pasos en el conocimiento de la estereopsis y la visión binocular, percatándose de que cada ojo percibe los objetos de forma diferente. Similares conclusiones alcanzaría Galenus cinco siglos más tarde.[33]

El siguiente hecho del que tenemos constancia se produce mucho más adelante en el tiempo con Leonardo da Vinci (1519) el cual avanzó en el estudio de la visión binocular, afirmando que es la culpable de la sensación de relieve que tienen los objetos[16].

Más adelante tenemos tímidos avances teóricos en este campo (con las investigaciones de Johann Kepler o Robert Smith entre otros), pero no fue hasta el siglo XIX cuando se inventó el primer dispositivo que aplicaba estos principios: el estereoscopio. Fue inventado por Sir Charles Wheatstone en 1840, quien además avanza profundamente en el conocimiento de la estereopsis y la visión binocular[32].

El estereoscopio es un dispositivo muy sencillo formado por dos imágenes (una para cada ojo), cuatro espejos para desviar las imágenes hacia los ojos y dos lentes, de modo que cuando se mira a través de ellas se forma en el cerebro una imagen tridimensional a partir de las dos imágenes. Podría considerarse el primer *headset* de realidad virtual.

El siguiente hito lo encontramos en el año 1935, con el escritor de ciencia ficción Stanley G. Weinbaum y su obra *Pygmalion's spectacles*. Este relato se desarrolla interactivamente con la participación del lector, pero lo realmente significativo es que gira en torno a unas gafas que recrean un mundo ficticio para aquel que las utiliza[25]. Por ello es considerado como el primer registro literario de la realidad virtual, y quien sabe si pudo servir de inspiración para futuras creaciones.

Posteriormente, en 1957, se produce la invención del primer «mundo virtual», el *Sensorama*, creado por Morton Heilig, considerado como el padre de la realidad virtual. El *Sensorama* es una máquina que pretende la inmersión multi-sensorial del usuario. Para ello utiliza un estereoscopio, un sistema de sonido estéreo, emisores de olores y un asiento móvil para proporcionar una experiencia inmersiva[17]. De este modo el espectador podía experimentar un viaje en motocicleta por las calles de Nueva York, sintiendo el viento generado por un ventilador, los olores de la gasolina y la comida de los puestos recreados por sustancias químicas, el movimiento de la silla, etc., elementos todos ellos generados en el momento oportuno de la reproducción del vídeo.

Tres años más tarde, en 1960, Heilig vuelve a ser protagonista en esta serie de hitos pues inventa las que se consideran como las primeras gafas de realidad virtual: el *aparato de televisión-estereoscópica para uso individual* también conocido como «Máscara Telesférica» [18]. Estas proporcionaban una sensación de inmersión en un mundo 3D a través de los sentidos de la vista y el oído.

Más tarde, en 1961 tendríamos la primera aplicación de esta tecnología en el campo militar, cuando los ingenieros de *Philco Corporation* desarrollaron el *Headsight*, un dispositivo para mostrar la imagen recogida por circuitos de cámara cerrada y así poder avistar amenazas desde una posición segura[28].

A comienzos de los años 70, *General Electric Company* desarrolló el que sería el primer simulador de vuelo con generación de imágenes por ordenador en tiempo real. Fueron utilizados por la NASA y por la armada de Estados Unidos[6].

Más tarde, en el 1975, se creó la primera plataforma interactiva de realidad virtual que no necesitaba de un HMD (*Head-Mounted Display*[1]) o guantes para interactuar con ella. Se trata del *Videoplace* de Myron kreuger's. Usaba proyectores, cámaras de vídeo y demás hardware específico para recoger las siluetas de los usuarios y proyectarlas en el entorno de realidad virtual[14].

En los siguientes años se van a producir numerosos avances sobre todo en el campo de la simulación de vuelo y el ámbito militar[29]:

- El casco *HMD VITAL* en 1979 que sigue los ojos de los pilotos y genera imágenes asociadas por ordenador, resultado de la investigación de la *McDonnell-Douglas Corporatioon.*
- El *Visually Coupled Airbone Systems Simulator* en el mismo año, otro simulador de vuelo de uso militar creado por Thomas Furness.
- En 1989 Furness mejoró su simulador con la «Super Cabina» *(Super Cockpit)*, en el que el piloto podía controlarlo a través de movimientos con los ojos, la voz, gestos y contaba con multitud de sensores, radares, mapas 3D, etc. Una versión análoga fue creada por la *Brithish Aerospace* basándose en el desarrollo del *Super Cockpit.*

Más adelante, en 1990 Jonathan Waldern presenta *Virtuality*, el primer juego de arcade basado en realidad virtual, en la exhibición de Gráficos por Computador en Londres[31].

Posteriormente, también en el ámbito de los videojuegos, Sega anuncia la creación de unas gafas de realidad virtual para la consola *Sega Genesis*, pero las dificultades técnicas impidieron que este proyecto siguiera adelante.

En 1997 se lanza otro proyecto interesante basado en realidad virtual, *Virtual Vietnam*, para ayudar en el tratamiento de estrés postraumático. Recreaba entornos bélicos a través de un HMD, como el interior de un helicóptero [12].

[1]Se traduce comúnmente por casco o dispositivo de realidad mixta, extendida o virtual.

Llegando al final de esta sección, ya entrado el siglo XXI se produce uno de los eventos más significativos en el campo de la realidad aumentada: el lanzamiento de la campaña *Kickstarter* de las gafas *Oculus Rift*, por Palmer Luckey en 2012, con tan solo 18 años. El primer prototipo ofrecía un campo de visión de 90º y dependía de la potencia de procesamiento de un ordenador para enviar las imágenes. Esto fue todo un revulsivo en el campo de la realidad virtual, aumentando el interés de las compañías por esta tecnología. Tanto interés despertó que Facebook acabó comprando la compañía *Oculus VR* por 2 billones de dólares.

Posteriormente numerosas compañías se sumaron a la investigación y lanzamiento de sus dispositivos de realidad extendida[26]:

- Sony con su *Project Morpheus*, un *headset* de realidad virtual para la Playstation 4.
- Google con su *Cardboard*, un modelo *low cost* de visor estereoscópico para smartphone.
- Samsung con su *Samsung Gear VR*, un *headset* que usa un *Samsung Galaxy* para mostrar las imágenes.
- HTC con su *VIVE SteamVR headset*, el primer dispositivo comercial con sensores que registran el movimiento del usuario y permiten moverse libremente en el espacio.
- Microsoft con sus *Hololens*, uno de los primeros dispositivos de auténtica realidad mixta.

2.2.2. Antecedentes en el campo de la ingeniería

El sector industrial es uno de los más prometedores donde la realidad mixta puede ser empleada para mejorar las técnicas actuales y proporcionar soluciones a problemas comunes[34].

En este sector, la realidad aumentada puede proporcionar al usuario una forma intuitiva de interactuar directamente con la información requerida para los procesos de la industria en cuestión. Las actividades en las que más utilidad ha aportado la realidad extendida es el de ensamblado, formación (entrenamiento, capacitación) y mantenimiento.

Por la temática del presente TFG, se centrará el estudio de los antecedentes de la realidad mixta en el campo de la ingeniería concretamente en los sectores del mantenimiento/reparación y la diagnosis/inspección, que son los que tienen más relación con la plataforma *Apholo*.

Uno de los campos dentro de los procesos industriales en el que la realidad extendida ha aportado y tiene mucho que aportar es el de mantenimiento, reparación y diagnóstico. A continuación se expondrán algunos proyectos y aplicaciones relevantes en este sector por orden cronológico.

Como proyecto pionero en esta temática destaca *Karma*, por Feiner y colaboradores en 1993, siglas de *Knowledge-based Augmented Reality for Maintenance Assistance*, el cual muestra instrucciones de mantenimiento y reparación de una impresora láser enfrente del usuario usando un casco de realidad aumentada[10]. De esta forma se conseguía una mejora de la eficiencia al no tener el usuario que depender de un ordenador o manual externo, pudiendo agilizar las tareas de mantenimiento y reparación.

Un año más tarde tenemos la aplicación *Grasp*, de Klaus Ahlers y colaboradores [1], escrita su parte software en C++ y que pretendía dar soporte a diferentes usos industriales. En el caso del mantenimiento y reparación se empleó para una aplicación que identificaba las partes de un complejo motor para efectuar reparaciones mecánicas.

En terreno militar tenemos también usos de la realidad virtual con objetivos de mantenimiento y reparación. Es el caso del sistema de *Urban*, quién desarrolló con su grupo de investigación un dispositivo óptico que, conectado a un chaleco *wireless*, podía mostrar manuales de reparación e imágenes de equipamiento durante maniobras militares o en el campo de batalla [30].

Dos años más tardes se publica un trabajo elaborado por Ockerman y colaboradores que expone la problemática típica de los técnicos de mantenimiento que tienen que estar desplazándose para realizar su labor y que, por lo tanto, resulta poco interesante el uso de un ordenador de mesa para guiar las labores de mantenimiento[20]. Este trabajo resulta interesante porque es una de las primeras veces

en las que se aborda de forma directa esta problemática aplicando un dispositivo de realidad virtual para no limitar la movilidad, el cual es uno de los motivos fundamentales a la hora de emprender proyectos de este tipo, como es el caso del proyecto *Apholo*.

En este estudio se observó que no se empeoraba el desempeño respecto a utilizar un libro y que mejoraba cuando las tareas eran de complejidad considerable. Resultados similares arrojó el trabajo publicado por Chung y colaboradores en 1999, donde compararon el desempeño a la hora de realizar una serie de tareas que requieren del uso de un manual de soporte. Se vio como el uso de un dispositivo de realidad aumentada mejoró los marcadores de eficiencia reduciendo el tiempo empleado para realizar las actividades respecto al uso de métodos tradicionales como un manual o un ordenador[8].

Un año más tarde, en 1998, Lipson y colaboradores presentan otro proyecto de realidad aumentada destinado a tareas de mantenimiento que requieren de una gran inversión en formación de los técnicos, como pueden ser aviones y vehículos grandes, equipamiento médico y plantas de producción[11]. El sistema presentado en esta publicación se compone de los siguientes elementos:

1. Programas de mantenimiento preconfigurados en forma de secuencias 3D multimedia.
2. Un enlace con el servidor web que proporciona los programas de mantenimiento requeridos.
3. Un dispositivo de realidad aumentada para mostrar al usuario los procedimientos de mantenimiento y que permite la interacción con el usuario.
4. Sensores opcionales en los ítems objeto de mantenimiento para mejorar la identificación.

Como observamos guarda muchas similitudes con el proyecto que hemos desarrollado para *HoloLens 2*, como por ejemplo las peticiones al servidor web (2) para obtener los procedimientos (1) y mostrarlos en el entorno a través del visor de las gafas de realidad mixta (3).

Otro antecedente en este campo lo representa un proyecto de investigación presentado por Neumann y Anthony en 1998, dónde se investigaban las posibilidades de la realidad aumentada con un sistema que renderizaba instrucciones contextuales de mantenimiento y reparación de componentes de una cabina de avión[23], necesitando marcadores para identificar los objetos por parte del sistema.

También, en 1998 se publica otro trabajo un tanto original, por parte de Lawson y Pretlove. Presentaron un sistema basado en realidad virtual cuyo objetivo era el de realizar mantenimiento en entornos peligrosos[15]. En este caso, el casco de realidad virtual recibe el input de información que recoge un vehículo robótico encargado de recorrer tuberías subterráneas, y el robot es controlado mediante los movimientos de la cabeza del usuario y un *jostick*. Las imágenes recibidas son mejoradas por el sistema aportando información superpuesta, mejorando el color, reconociendo y destacando averías, etc.

Posteriormente, en 1999 se encuentra la publicación de un proyecto que incorpora novedades. Fue publicado por Sundareswaran y colaboradores, y alguno de los puntos novedosos que incorpora es el manejo del sistema a través de comandos de voz[2], una forma de interacción que se ha implementado también en la plataforma *Apholo*. El objetivo de este sistema es la diagnosis de dispositivos electrónicos complejos, tales como ordenadores personales. El sistema combina realidad virtual con realidad aumentada (el término realidad mixta todavía no se utilizaba pero podría aplicarse a este proyecto), al renderizar elementos 3D a la par que superpone información en el entorno real del usuario.

Hasta ahora la mayoría de proyectos se habían realizado en cascos de realidad aumentada, más rudimentarios que los del siglo XXI, pero con la misma esencia. En este sentido supone una novedad el trabajo publicado por Nakagawa y colaboradores en 1999, en el que se reflejan problemas de usabilidad de los típicos cascos de realidad aumentada [21]. Para intentar solventarlos se presentaba un prototipo para el soporte de técnicos de mantenimiento de plantas de producción de energía basado en realidad virtual, pero con la particularidad de no estar aplicado en unas gafas o casco de realidad virtual sino en un ordenador de mano, como una forma más suave y progresiva de realizar

la transición al típico «HMD», debido a las quejas reportadas por los técnicos sobre la fatiga que les generaba.

Pasando ya a la siguiente centuria, otro antecedente se encuentra en un trabajo publicado por Klinker y colaboradores que presenta un proyecto realizado por estudiantes de último curso de Ingeniería del Software basado en realidad aumentada bautizado como STARS *(Sticky Technology for Augmented Reality Systems*[13]. La función principal del sistema era asistir a los técnicos de mantenimiento y reparación de centrales nucleares. Para ello, el sistema reconocía el estado actual en el proceso de reparación y mostraba las instrucciones oportunas para continuar con el procedimiento.

Siguiendo con la temática de las centrales eléctricas, en el año 2003 Chikahito and Norihiko presentaron un proyecto basado en realidad aumentada para realizar el mantenimiento de las mismas[22]. Uno de los principales desafíos y novedades presentes en este trabajo fue la identificación de los componentes objeto de mantenimiento sin necesidad de utilizar marcadores específicos que facilitasen la tarea.

Otro ejemplo de aplicación de la AR, más enfocado al ámbito de la formación para el mantenimiento, viene representado por el trabajo publicado por Boulanger en 2004[4]. En el se expone un sistema destinado a la teleformación de técnicos para labores de reparación. Habitualmente esta tarea se realizaba a distancia a través de llamadas telefónicas pero este canal presenta muchas deficiencias para la enseñanza de una tarea donde el componente visual es muy importante. Este proyecto por lo tanto proponía guiar los procesos de reparación a través de la realidad virtual mediante instrucciones, ayuda contextual reconociendo elementos con marcadores, además de mantener la asistencia telefónica que ya venían utilizando incorporándola a este sistema.

Por último, cabe destacar el trabajo de Wojciech Moczulski y colaboradores *Applications of augmented reality in machinery design, maintenance and diagnostics*[19]. En esta publicación se presentan tres aplicaciones para mostrar posibles usos de la realidad aumentada dentro del campo de la ingeniería industrial, concretamente para el diseño, el mantenimiento y el diagnóstico de maquinaria. Las que más nos interesan por la temática del presente TFG son las de mantenimiento y diagnosis.

- Respecto a la de **mantenimiento**, se plantea la identificación de los objetos (denominados EGD, *Electronic Gaming Devices*) sobre los que realizar mantenimiento mediante una cámara y una base de datos. Posteriormente las imágenes serían analizadas y se transmitirían instrucciones al usuario para realizar el mantenimiento.
- La aplicación de **diagnosis** está destinada a ayudar al técnico a medir niveles de ruido alrededor de una máquina y a realizar el diagnóstico pertinente. El sistema estaría compuesto por una cámara USB, un monitor, un ordenador, un marcador y la biblioteca de tracking *ARToolKit* para Matlab. La aplicación indicaría cual es la posición idónea desde la que realizar la medición, y posteriormente, con los datos obtenidos de la medición, el sistema los transformaría en información valiosa mostrando círculos más o menos intensos que ayudarían al técnico en la elaboración del diagnóstico.

Por último, para tener una visión de conjunto de la investigación realizada en los últimos años, adjunto un par de tablas extraídas de la publicación de Eleonora Bottani y Giuseppe Vignali[3] que recogen los estudios con aplicación que se han publicado desde el 2006 al 2016 clasificados por campo de aplicación y por sector industrial.

Tabla 2.1: Publicaciones sobre realidad aumentada con aplicación entre los años 2006-2016 clasificados por campo de aplicación

Campo de aplicación	Publicaciones
Ensamblado	15
Mantenimiento	23
Diseño de producto	11
Seguridad	8
Asistencia remota	6
Robótica	1
Ergonomía	1
Formación	13
Control de calidad	3
Inspección o gestión de instalaciones	4
Entorno exterior	1
Picking	3
Diagnóstico	3
Prototipado	1
Información	1
Navegación	1
2D/3D CAD	1
Planificación de diseño	1
Soldadura	1
Simulación	1

Tabla 2.2: Publicaciones sobre realidad aumentada con aplicación entre los años 2006-2016 clasificados por sector industrial

Sector industrial	Publicaciones
Aviación	3
AECO	5
Automoción	6
Centrales químicas	1
Electrónica	4
Calzado	2
Laboratorio	23
Herramientas para maquinaria	7
Manufactura	9
Almacenamiento	4
Centrales de energía	4
Otros	1

CAPÍTULO 3
Objetivo

3.1. OBJETIVO GENERAL

El objetivo principal del proyecto *Apholo* es la construcción de una plataforma software para el desarrollo de aplicaciones de realidad mixta para *HoloLens 2* que facilite la convergencia tecnológica con aplicaciones existentes de ámbito industrial.

Esta plataforma software estará formada por un conjunto de tres modelos de aplicación destinadas a posibilitar la conversión y/o convergencia de aplicaciones existentes del sector industrial con las tecnología de la realidad mixta, constituyendo el dispositivo soporte de esta tecnología las gafas *HoloLens 2* de Microsoft.

3.2. OBJETIVOS ESPECÍFICOS

3.2.1. Usabilidad: el sistema deberá ser usable

La usabilidad, entendida como *la capacidad de un software de ser comprendido, aprendido, usado y ser atractivo para el usuario, en condiciones específicas de uso* según la norma ISO 9126[1], debe ser un objetivo presente en cualquier proyecto software, pero lo es todavía más si cabe en este pues, el introducir formas de interacción tan novedosas como las ofrecidas por los dispositivos de realidad mixta, puede suponer para los nuevos usuarios un choque demasiado brusco respecto a su modo habitual de uso de aplicaciones informáticas.

Es por ello que, en este proyecto, se pondrá especial énfasis en buscar una buena usabilidad que facilite el rápido aprendizaje, comprensión y la facilidad de uso para los usuarios que se inicien en el uso de este tipo de aplicaciones.

3.2.2. Empleo de diferentes canales de interacción: la interacción con el sistema deberá basarse en gestos y comandos de voz

Una de las potenciales ventajas de las tecnologías de la realidad extendida es el uso de novedosos canales de interacción que ofrezcan innovaciones y mejoras en cuanto a la usabilidad y la experiencia del usuario respecto a los mecanismos tradicionales.

El uso de la realidad mixta en este proyecto pretende aportar una mejora en este sentido respecto a las aplicaciones tradicionales de escritorio. Es por ello que se apostará por la utilización de los siguientes canales de interacción en la plataforma *Apholo*:

1. Interacción gestual, ya sea por contacto directo o a distancia.
2. Interacción por comandos de voz.

[1] https://www.une.org/encuentra-tu-norma/busca-tu-norma/norma/?c=N0032555

3.2.3. Mantenibilidad y escalabilidad: el sistema deberá construirse atendiendo a criterios de mantenibilidad y escalabilidad

El proyecto debe ser diseñado y construido atendiendo a criterios que garanticen una buena mantenibilidad y escalabilidad, ya no solo porque son atributos deseables y necesarios en cualquier proyecto software, sino además porque el ciclo de vida del sistema *Apholo* dependerá del ciclo de vida de las aplicaciones de ámbito industrial a las cuales vaya asociado (la parte final de uso y mantenimiento), por lo que debe ser capaz de crecer en ritmo y dimensiones similares a los de estas aplicaciones asociadas.

3.2.4. Sistema plurilingüe: el sistema deberá soportar varios idiomas

El sistema deberá soportar el uso de diferentes idiomas, para así asegurar la compatibilidad en este sentido con aplicaciones de ámbito industrial que utilicen diferentes idiomas, lo cual es habitual.

3.2.5. Delegación de tareas y comunicación: el sistema deberá desarrollarse siguiendo una arquitectura cliente-servidor

La arquitectura empleada por el sistema en el módulo 3 será del tipo cliente-servidor, tratándose del cliente el dispositivo HoloLens 2 y el servidor el computador anfitrión de la aplicación de escritorio. De este modo se intentará delegar en el computador el mayor cómputo de operaciones posibles, para liberar en la medida de lo posible al dispositivo de realidad mixta, el cual dispone de una reducida capacidad de procesamiento.

3.2.6. Mantenibilidad y escalabilidad: el sistema deberá construirse atendiendo a criterios de mantenibilidad y escalabilidad

El proyecto debe ser diseñado y construido atendiendo a criterios que garanticen una buena mantenibilidad y escalabilidad, ya no solo porque son atributos deseables y necesarios en cualquier proyecto software, sino además el ciclo de vida del sistema *Apholo* dependerá del ciclo de vida de las aplicaciones de ámbito industrial a las cuales vaya asociado (la parte final de uso y mantenimiento), por lo que debe ser capaz de crecer en ritmo y dimensiones similares a los de estas aplicaciones asociadas.

3.2.7. Estabilidad del sistema y recuperación de errores: el sistema deberá ser capaz de recuperarse ante errores

Se asegurará que todas las aplicaciones de esta plataforma puedan recuperarse ante posibles errores e incidencias, continuando su funcionamiento adecuadamente y evitando cierres bruscos y estados de no retorno.

3.2.8. Uso de tecnologías y estándares libres: el sistema deberá ser desarrollado mediante el uso de herramientas libres

El proyecto será desarrollado mediante el uso de *frameworks*, tecnologías y estándares libres en la medida de lo posible, de tal modo que se facilite su accesibilidad, mantenibilidad y funcionamiento en el futuro.

3.2.9. Alta optimización: el sistema deberá estar fuertemente optimizado

En este proyecto la optimización del procesamiento y el cómputo es de vital importancia debido a las limitaciones técnicas que imponen el casco de realidad mixta *HoloLens 2*.

Por lo tanto, será de vital importancia el uso del computador para delegar en él el procesamiento de las tareas más pesadas y gestionar correctamente la comunicación entre ambas aplicaciones para evitar cuellos de botella y agilizar los procesos.

CAPÍTULO 4

Metodología

4.1. MÉTODO DE TRABAJO

Antes de nada es necesario remarcar que este proyecto se ha realizado trabajando en un equipo de desarrollo formado por dos personas, aunque debido a que se trata de un Trabajo Fin de Grado la mayor parte de la dedicación corresponde a la realizada por el alumno que presenta este TFG (se verá reflejado en la captura de *commits* que se adjunta en el capítulo de conclusiones 6).

A continuación se expondrán los elementos principales del método de trabajo empleado para el desarrollo del presente TFG.

4.1.1. Metodología de desarrollo

Para el desarrollo del proyecto se ha seguido una metodología de tipo RAD (*Rapid Application Development*, Desarrollo Rápido de Aplicaciones).

Esta metodología da respuesta a las necesidades que planteaba el proyecto, pues ha sido de gran valor el contar con prototipos tanto no funcionales como funcionales en etapas tempranas del ciclo de desarrollo para asegurar una correcta elicitación de requisitos y avanzar así en la dirección correcta.

Así pues, en las diferentes etapas del proyecto, se ha comenzado cada una con unos requisitos iniciales, a partir de los cuales se procedió a la construcción de prototipos no funcionales en el menor tiempo posible para mostrárselos al cliente y terminar de refinar los requisitos. Con los requisitos finales elicitados, el siguiente objetivo que se marca es el de la construcción de un prototipo con una parte representativa de la funcionalidad del sistema para que, una vez construido, se enseñe de nuevo al cliente en el menor tiempo posible. Y así en sucesivos ciclos.

De este modo, el sistema va refinándose, obteniendo al final de cada iteración una versión más desarrollada que mostramos al cliente, el cual puede ver y garantizar que el proyecto está siendo desarrollado en la dirección correcta.

4.1.2. Gestión de las ramas del repositorio (*branching*)

También es conveniente destacar la metodología usada para el branching o gestión de las ramas del repositorio, basado en *GitFlow*. La organización de ramas que hemos seguido, por tanto, ha sido la siguiente:

- **Master**: Almacena el historial de publicación oficial.
- **Dev**: Rama de integración de las ramas de funcionalidad o *features*
- **Feature**: Ramas para trabajar en una funcionalidad determinada.
- **HotFix**: Ramas destinadas a la corrección de errores.

4.1.3. Reuniones

Otra de las bases del método de trabajo han sido las reuniones periódicas, de tres tipos fundamentalmente:

A. **Con el *product owner*** (en términos de *Scrum*), **de periodicidad bisemanal**. En estas reuniones se nos comunicaba los nuevos requisitos y modificaciones que teníamos que hacer al sistema y se definían los siguientes hitos principales y las fechas clave.

B. **Con mi compañero de trabajo, de periodicidad bisemanal**. Estas reuniones se realizaban con posterioridad a la reunión con el *product owner* para definir las nuevas tareas y tarjetas que íbamos a añadir al tablero Canvas.

C. **Con mi compañero de trabajo, de periodicidad diaria**. En estas reuniones le comunicaba los avances realizados el día anterior, los problemas que me habían ido surgiendo y concretábamos el plan de trabajo para ese día.

4.1.4. Gestión de tareas

Al trabajar en equipo cobra mayor relevancia la organización de las tareas para alcanzar los hitos establecidos. Para ello se ha utilizado un tablero de tipo Kanban con la siguiente distribución de columnas:

- **Columna de *sprints***: en estas columnas se almacenan todas las tareas relativas a un sprint al inicio del mismo.
- **Columna "*To-Do*"**: recoge las tareas que requieren hacerse a continuación dentro de un sprint.
- **Columna "*In progress*"**: muestra las tareas que están siendo llevadas a cabo en ese momento.
- **Columna "*Done*"**: refleja las tareas que han sido completadas.
- **Columna "*On hold*"**: almacena las tareas que no pueden iniciarse en este momento por dependencia con otras tareas o por otros motivos.

Para la gestión del tablero se ha utilizado la herramienta *Trello*[1].

4.1.5. Modalidad del trabajo

Otro punto destacable respecto a la metodología es que se ha seguido fundamentalmente una modalidad de trabajo mixta compaginando teletrabajo y presencialidad. Cabe destacar que la comunicación entre los miembros del proyecto ha sido muy fluida siempre gracias al uso de diversas herramientas de comunicación que se detallarán en posteriores epígrafes.

4.2. PATRONES DE DISEÑO

A continuación se destacarán algunos patrones de diseño utilizados en la construcción del sistema *Apholo*.

4.2.1. Patrón componente

Se trata de uno de los principales patrones de este proyecto. Es descrito en el libro *Game Programming Patterns*[24] y es un patrón asociado fuertemente al motor de desarrollo de videojuegos Unity.

La característica fundamental de este patrón es que las entidades en un proyecto son meros contenedores sin funcionalidad propia. A estos contenedores se le deben añadir componentes, que son *scripts* que aportan funcionalidad a ese contenedor.

Por ejemplo, el componente «interactable» permite una serie de interacciones con el contenedor (*GameObject*) al que se lo añadamos. Por lo tanto, si queremos que un objeto tenga ese comportamiento,

[1] https://trello.com/

tan solo tendremos que añadirle este *script*. En este proyecto, este componente «interactable» lo llevan objetos como botones y ventanas, ya que ambos deben permitir la interacción con ellos.

4.2.2. Patrón *listener*

También conocido como *Observer* o *Event-subscriber*, es un patrón que permite implementar un mecanismo de suscripción mediante el cual un objeto actúa como «avisador» y el resto se suscribe al mismo convirtiéndose en los «oyentes». De este modo, cuando suceda el cambio que queremos notificar, el objeto avisador notificará a los objetos que le estén escuchando de que se ha producido dicho cambio.

Este patrón ha sido verdaderamente útil en este proyecto de cara a gestionar diversos flujos de información, como las peticiones API al servidor web, ya que, una vez enviada la petición, se ha de esperar la respuesta del servidor con la información solicitada. Entonces, cuando esta respuesta es recibida, el objeto encargado de recibirla debe avisar a los demás objetos que están esperando esta respuesta para procesar los datos recibidos y continuar la cadena de procesamiento.

Otra situación en la que se ha aplicado este evento es de cara a comunicar la aplicación de escritorio con la aplicación de *HoloLens 2*. Ya sea en el módulo de *streaming* o en el nativo, es necesario enviar información de un sistema al otro bidireccionalmente, y para ello deben haber objetos que estén a la escucha de estos mensajes. Así pues, cuando dichos objetos avisadores reciben un mensaje de la otra aplicación, notifican a los componentes que están suscritos para que continúen con las tareas que tienen encomendadas.

4.2.3. Event Queue Patter

Es un patrón parecido al patrón evento-suscriptor, pero tiene alguna peculiaridad. Básicamente consiste en un esquema similar al de avisador-oyentes, pero añadiendo entre ambos una cola donde se van almacenando las notificaciones. Esto es especialmente útil cuando el tiempo de procesamiento de las tareas por parte de los oyentes es relativamente largo, ya que si todavía está procesando una, no puede atender a nuevos avisos. Con este método, los eventos van guardándose en la cola y los oyentes pueden procesarlos según van terminando sus tareas.

Un ejemplo de aplicación de este patrón en el proyecto ha sido cuando se han tenido que enviar eventos de interacción en la aplicación de *streaming*. Esta forma de gestionar los eventos mejora el comportamiento general de la aplicación, pero especialmente con el evento de "pulsar y arrastrar", ya que requería de cierto tiempo de procesamiento debido a su propia naturaleza, por lo que para evitar solapamientos era necesario el envío de las pulsaciones a esta cola y dejar que el ordenador anfitrión los procesara de forma consecutiva.

4.2.4. Flyweight pattern

Este patrón consiste en la reutilización de variables para el máximo número de objetos posibles que compartan el valor de esa variable en vez de inicializar una nueva variable para cada uno de estos objetos. Esto es interesante ya que inicializando menos variables se ahorra memoria y tiempo de procesamiento.

Un caso de aplicación entre muchos lo tenemos por ejemplo al instanciar un objeto de un botón en un componente que va a crear botones de ese tipo. En vez de llamar al método que busca ese objeto cada vez que lo necesitamos instanciar, se instancia una vez al principio del *script* y se utiliza esa variable para instanciar los demás.

4.2.5. Patrón estado

Este patrón permite modificar el comportamiento de un objeto en función del estado en el que se encuentre. De este modo un objeto se puede comportar de formas diferentes.

Este patrón se ha aplicado por ejemplo a la hora de configurar los botones de tipo *toggle*, es decir, con estados de encendido y apagado. En este ejemplo está claro que dependiendo del estado, el botón realizará unas funciones u otras al ser pulsado.

4.3. MEDIOS UTILIZADOS DURANTE EL DESARROLLO

A continuación, se expondrán los medios tanto hardware como software utilizados para desarrollar este proyecto.

4.3.1. Medios Hardware

Los medios de tipo hardware que se han empleado para el desarrollo del proyecto Apholo han sido los siguientes:

- **Ordenador portátil Macbook Pro (15", Mediados 2016)**, utilizado principalmente en las etapas iniciales del desarrollo.
- **Ordenador de sobremesa HP ENVY Desktop** (TE01-1032ns Intel Core i7-10700F/32GB/1TB SSD/RTX 3060Ti). En este ordenador se ha desarrollado el proyecto casi en su totalidad.
- **Microsoft HoloLens 2.** Es el casco de realidad mixta que se ha empleado en el desarrollo del TFG, actual estándar de facto en la industria.

4.3.2. Medios Software

- **Windows 10 Home sobre Bootcamp**. Bootcamp es el gestor de particiones de Windows en ordenadores Apple, y fue necesario su uso para disponer del entorno de desarrollo adecuado para este proyecto ya que debía de ser Windows por temas de compatibilidad. Solo se usó al inicio del proyecto.
- **Windows 11 Home.** Cuando el proyecto pasó a desarrollarse en el ordenador de sobre mesa el sistema utilizado fue Windows 11 Home, que mantiene la compatibilidad de su predecesor.
- **Git.** El sistema de control de versiones empleado ha sido Git de cara a mantener un repositorio del proyecto, disponer de historial de cambios, gestión de ramas de trabajo, acceso centralizado, etc.
- **Bitbucket.** Junto a Git se ha usado el servicio de repositorios de Bitbucket ya que es el que utilizan de forma habitual en Furious Koalas Interactive, la empresa donde se han realizado las prácticas asociadas a este TFG.
- **GitHub Desktop.** Para gestionar el repositorio de forma rápida y sencilla se ha empleado Github Desktop, que presenta compatibilidad con Bitbucket.
- **Unity.** Plataforma de desarrollo en tiempo real y motor de videojuegos que se ha empleado para el desarrollo de este proyecto de realidad mixta.
- **Unity Hub.** Gestor de proyectos de Unity.
- **Microsoft Visual Studio Community 2019.6.** IDE principal utilizado para la fase de codificación o escritura de código del sistema.
- **Microsoft Visual Studio Code.** IDE secundario empleado para algunas tareas menores.
- **Aplicación de ámbito industrial.** Aplicación modelo sobre la que se ha basado el desarrollo del módulo nativo para HoloLens 2. Su nombre no se puede revelar por motivos de confidencialidad.
- **Slack.** Herramienta empleada para la comunicación en el entorno de trabajo de la empresa en la que se ha realizado el TFG y las prácticas asociadas.
- **Teams.** Aplicación utilizada para la comunicación con la empresa externa con la que se ha trabajado para el desarrollo del proyecto.

- **Cloc.** Herramienta empleada para obtener estadísticas de las líneas de código, comentarios y lenguajes empleados en los archivos de código fuente del proyecto.
- **Trello.** Trello es un software de administración de proyectos con interfaz web y con cliente para iOS y android para organizar proyectos. Se ha empleadopara la gestión de los sprints, hitos y la organización de las tareas siguiendo la metodología Kanban gestionando las tareas en columnas to-do, doing y done.

4.3.3. 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 ha empleado este lenguaje porque es el mejor integrado en el ecosistema de Unity y el que mejor soporte tiene.
- **Microsoft Mixed Reality Toolkit (MRTK)**[2]. Se trata de un proyecto de Micrososoft que consiste en una biblioteca de componentes y características para el rápido desarrollo de aplicaciones basadas en las tecnologías de la realidad mixta. Ha sido de gran utilidad para la reutilización de botones y componentes de la interfaz además de los múltiples componentes que añaden funcionalidad a los objetos.
- **Mixed Reality WebRTC**[3]**.** Se trata de un proyecto libre y de código abierto que proporciona a los navegadores web y a las aplicaciones móviles comunicación en tiempo real a través de interfaces de programación de aplicaciones. Se ha empleado para la comunicación entre la aplicación de escritorio y la aplicación de realidad mixta de HoloLens 2.

[2] https://docs.microsoft.com/es-es/windows/mixed-reality/mrtk-unity/mrtk2/?view=mrtkunity-2022-05/
[3] https://github.com/microsoft/MixedReality-WebRTC/

CAPÍTULO 5
Resultados

En este capítulo se expondrán los resultados alcanzados con la elaboración del TFG, destacando y explicando las partes más importantes de los diferentes módulos. Para organizar la exposición de los resultados se dividirá este capítulo en tres subepígrafes, uno por cada módulo que compone el sistema.

5.1. CONSIDERACIONES PREVIAS

5.1.1. Módulos de la plataforma Apholo

El objetivo de la plataforma Apholo es la facilitación del desarrollo de aplicaciones basadas en realidad mixta asociadas a aplicaciones de ámbito de industrial. En otras palabras, lo que se pretende es desarrollar una serie de soluciones (módulos) que permitan de forma rápida desarrollar, convertir o portar una aplicación tradicional (de pc, por ejemplo) a una basada en realidad mixta, en este caso para el dispositivo HoloLens 2.

De este modo, se idearon tres posibles soluciones o módulos que son los que se han desarrollado en el presente TFG. A continuación, se expone brevemente cada uno de ellos antes de pasar a sus epígrafes correspondientes.

A. **Módulo 1: Aplicación de captura de pantalla y streaming**. Esta aplicación permite la captura en tiempo real de la imagen de una aplicación que esté ejecutándose en un ordenador y su envío al dispositivo de realidad mixta HoloLens 2 para que sean mostrados en un componente de holograma que permite la interacción a través de gestos y comandos de voz. Esta interacción es traducida a eventos de ratón y teclado que inician acciones sobre la aplicación como si se estuviera usando de forma directa.

B. **Módulo 2: Aplicación de soporte remoto**. Este módulo por su parte lo que ofrece es una función de soporte remoto mediante video llamada, con una función de streaming semejante al módulo anterior. En este caso la imagen que se envía al holograma en la aplicación de las gafas es la recogida por una webcam típica, y a la persona que se encuentra en el ordenador le llegará la imagen recogida a través de la cámara de las HoloLens 2.

C. **Módulo 3: Aplicación con componentes nativos de HoloLens 2**.Este módulo constituye la solución más compleja y la más personalizada a cada aplicación, pues se basa en la construcción de una aplicación con componentes nativos de HoloLens 2 para desarrollar o portar una aplicación ya existente de ámbito industrial. Los componentes de esta aplicación utilizan scripts y estructuras fácilmente reutilizables de modo que puedan ser adaptadas con facilidad en múltiples proyectos.

5.1.2. Organización de proyectos en Unity

Es conveniente explicar someramente la organización de los proyectos en Unity, la plataforma utilizada para el desarrollo de los módulos. En Unity los proyectos se componen de escenas que son estructuras de datos las cuales almacenan instancias de los objetos (ya sean gráficos, scripts de funcionalidades, sonidos…) que componen el proyecto, y que tienen una configuración determinada para esa escena concreta.

Así pues, para el desarrolla de esta plataforma la estructura que se ha seguido es la división primera de la plataforma en dos proyectos: Uno que engloba la parte de los módulos de aplicación de PC, y el otro proyecto que engloba la parte de los módulos de la aplicación para las HoloLens 2.

De este modo, en cada proyecto tendríamos una escena para cada uno de los módulos. En el siguiente diagrama se representa esta organización general de la plataforma en Unity.

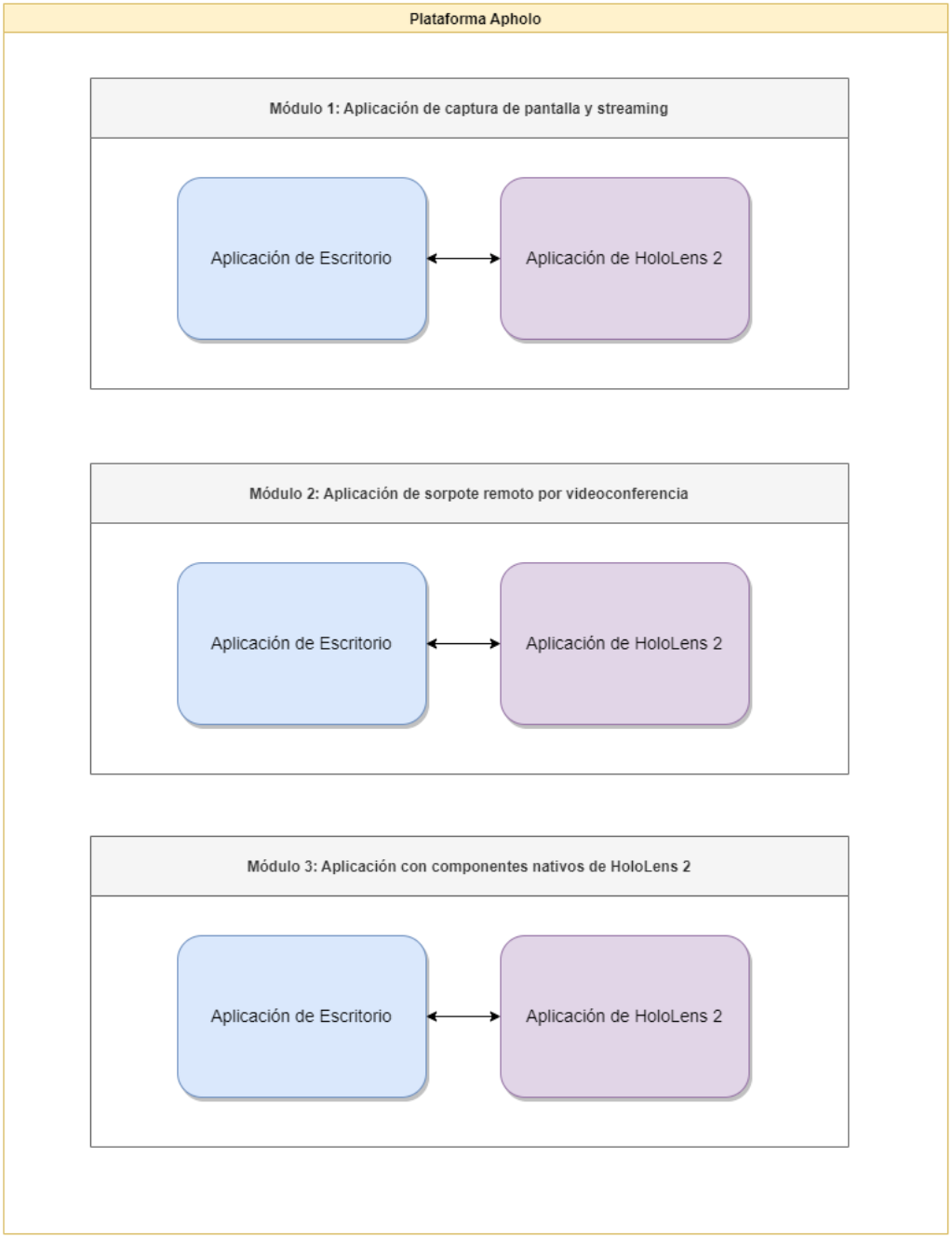

Figura 5.1: Organización de los proyectos de la plataforma Apholo en Unity

5.2. MÓDULO 1. APLICACIÓN DE CAPTURA DE PANTALLA Y STREAMING

Para empezar este apartado y los siguientes, se expondrá un diagrama general mostrando los subsistemas que componen cada módulo.

De este modo, en este primer módulo contamos la división bipartita que se repetirá en los siguientes módulos dividiendo cada uno en una aplicación de escritorio y otra para las HoloLens 2, cada una con sus funciones particulares.

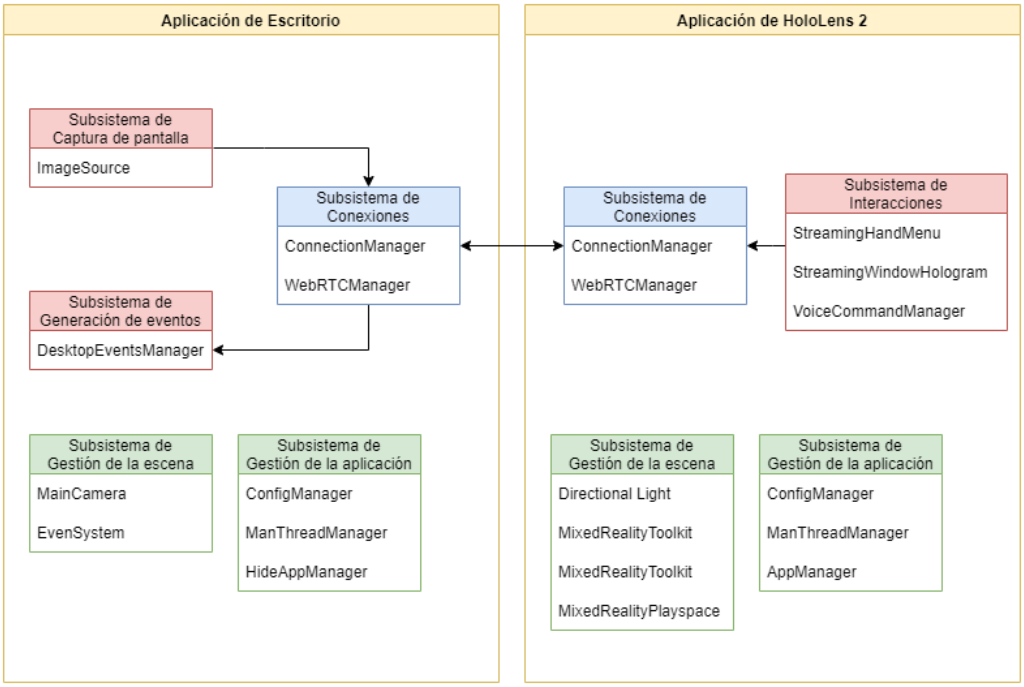

Figura 5.2: Subsistemas que componen la aplicación de captura de pantalla y streaming

Los subsistemas principales de la aplicación de escritorio son:

- **Subsistema de captura de pantalla**: es el encargado de captura la imagen de la ventana de la aplicación que queremos transmitir a las HoloLens 2 para que el usuario pueda interactuar con ella de forma remota.
- **Subsistema de conexiones**: es el responsable de establecer y gestionar la conexión con la aplicación de HoloLens 2 para transmitir la imagen y recibir la interacción que realice el usuario en la aplicación de HoloLens 2.
- **Subsistema de generación de eventos**: este subsistema tiene la función de traducir la interacción realizada por el usuario en eventos de teclado y ratón que permiten realizar acciones sobre la aplicación que se esté ejecutando en el ordenador, consiguiendo así la interacción de forma remota.
- **Subsistema de gestión de la aplicación**: subsistema encargado de algunas funciones generales como la gestión de un archivo de configuración para guardar y cargar atributos de configuración, gestionar el hilo principal de la aplicación (es el único que puede realizar ciertas acciones), o hacer que la aplicación se ejecute en segundo plano.
- **Subsistema de gestión de la escena**: subsistema con objetos predeterminados y propios de Unity para la gestión de la escena.

Por otra parte, los subsistemas más importantes de la aplicación de HoloLens 2 son:

- **Subsistema de interacciones**: subsistema responsable de la recogida y gestión de las interacciones que realice el usuario con la ventana holográfica que muestra la imagen de la aplicación de ordenador. También gestiona el componente «Menú Mano» propio de MRTK con botones para funciones rápidas, como la resposición de la ventana o la apertura del teclado holográfico entre otras.

- **Subsistema de conexiones**: componente análogo al de la aplicación de escritorio, con mismos objetos y funciones. Gestiona el canal por el que se transmitirán los eventos de interacción y se recibe la imagen *stremeada* de la aplicación de escritorio.

- **Subsistema de gestión de la aplicación**: mismas funciones que en el caso de la aplicación de escritorio incluyendo un componente «AppManager» encargado de gestionar el cierre de la ventana del holograma.

- **Subsistema de gestión de la escena**: comparte la funcionalidad con el mismo sistema de la aplicación de escritorio, pero incluye los componentes adicionales de MRTK para la gestión de proyectos de realidad mixta.

A continuación se muestra un diagrama de flujo para ayudar a la comprensión del funcionamiento de este *Módulo 1* de la plataforma *Apholo*.

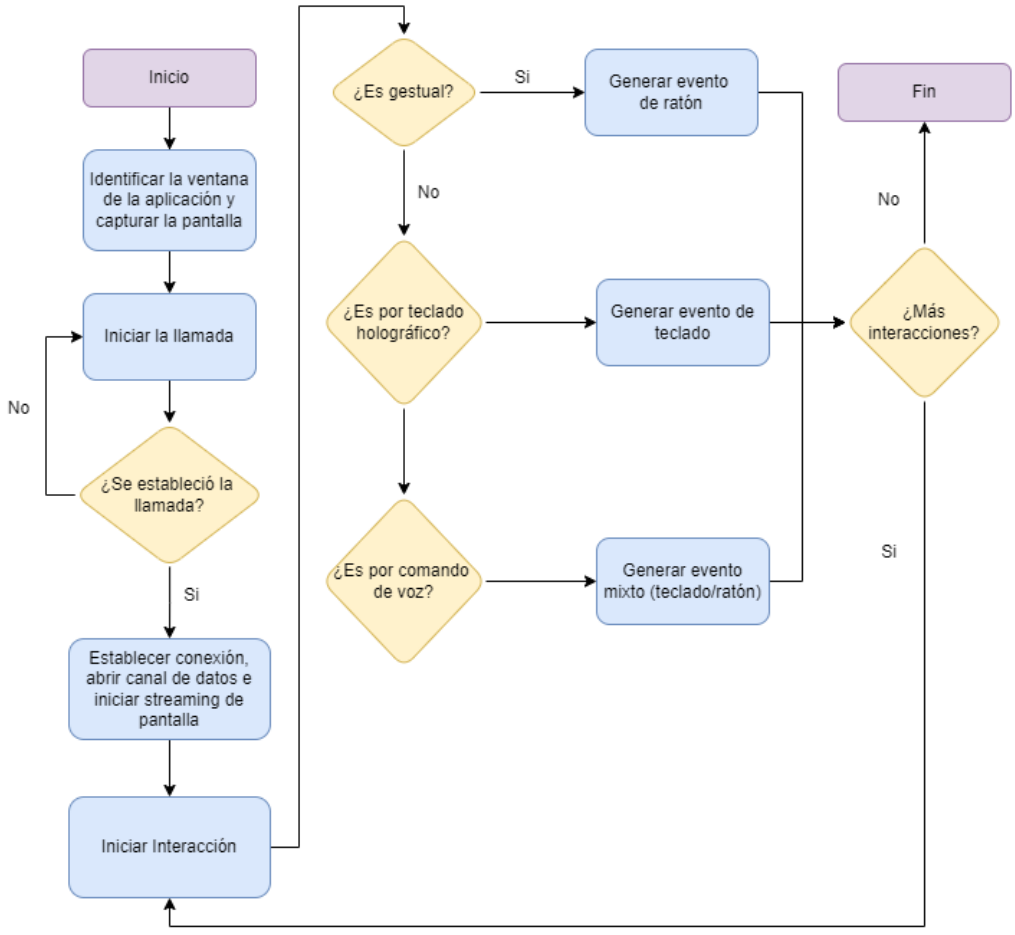

Figura 5.3: Diagrama de flujo del módulo 1

5.2.1. Aplicación de Escritorio

A continuación se muestra un diagrama de clases simplificado que expone los subsistemas expuestos previamente con sus clases, atributos, métodos y relaciones más significativas.

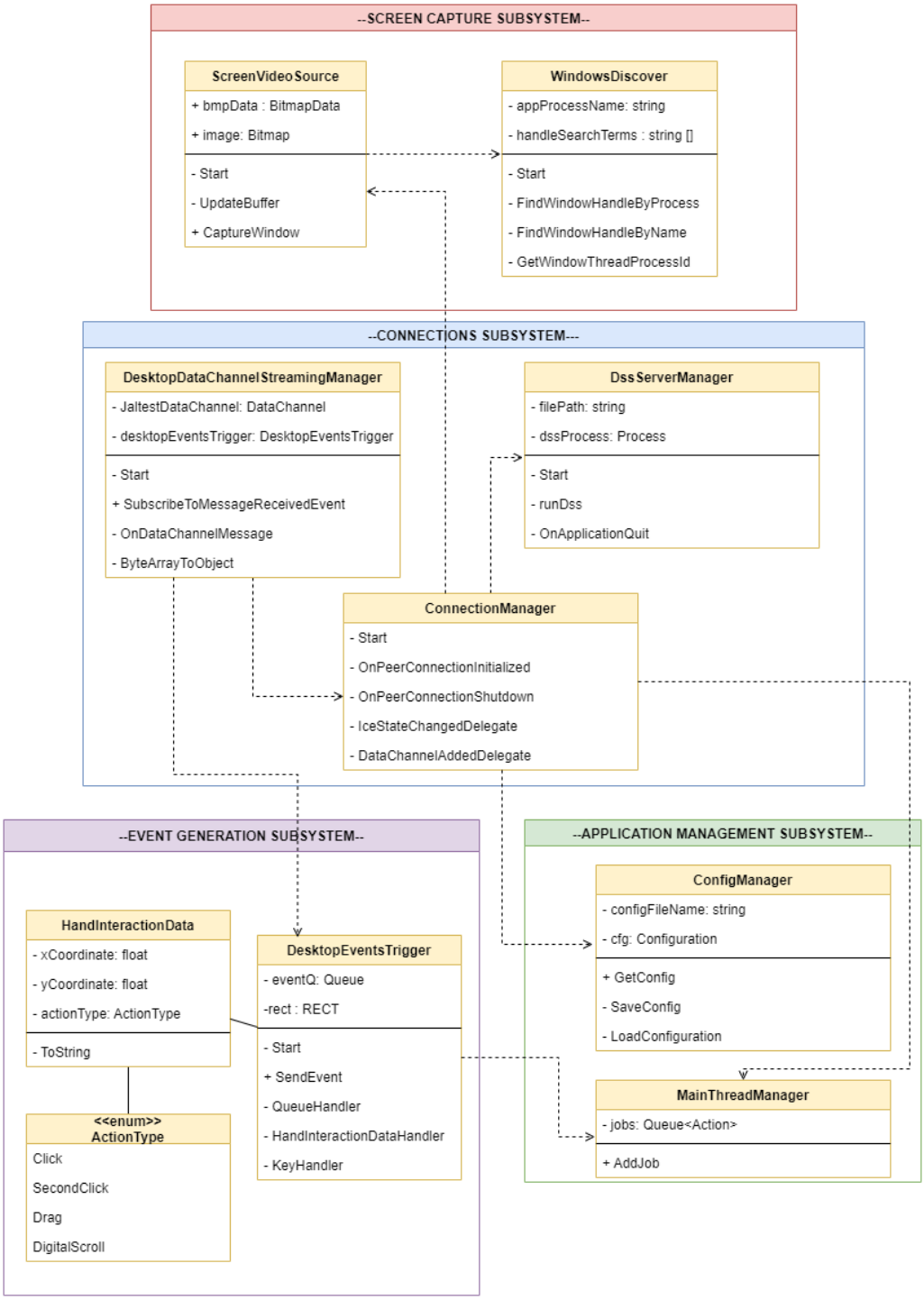

Figura 5.4: Diagrama de clases de la aplicación de escritorio del módulo 1)

La lógica que se ha seguido a la hora de agrupar los diferentes componentes de la escena es la agrupación por responsabilidades, es decir, agrupar en mismo objeto / GameObject los scripts que estén relacionados con una función / responsabilidad. Por ejemplo, todos los scripts que tengan que ver con la generación de eventos de ordenador se han agrupado en el objeto DesktopEventsManager.

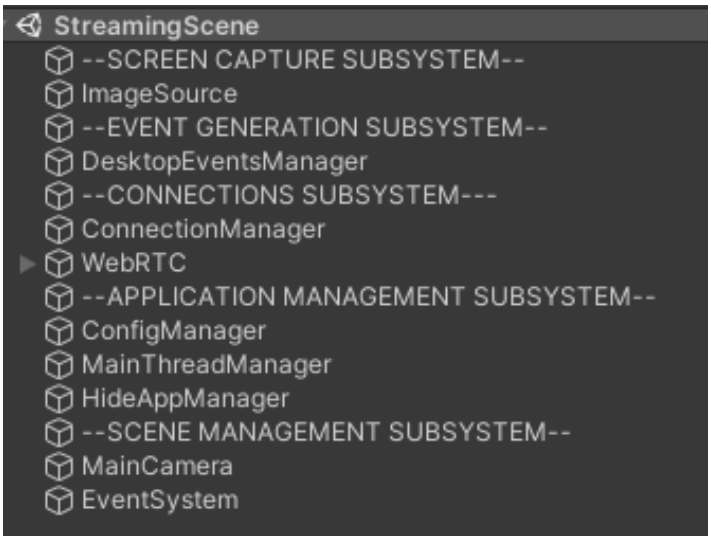

Figura 5.5: Componentes de la escena del módulo 1 en Unity usando GameObjects vacíos como separadores

Los tres grupos de objetos más importantes de esta escena y por tanto de esta aplicación son:

1) Componentes relativos a la generación de eventos de teclado y ratón en el ordenador.
2) Componentes relativos a la captura de pantalla.
3) Componentes relativos a la gestión de los mensajes del canal de datos y Mixed Reality WebRTC.

1) Componentes relativos a la generación de eventos de teclado y ratón en el ordenador

Una de las partes más importantes de esta aplicación es la relativa a la generación de eventos de teclado y ratón a partir de las interacciones que realice en el usuario en las *HoloLens 2*. Estos eventos se generan a partir de la clase *DesktopEventTrigger*, presente en el objeto de la escena *DesktopEventsManager*. Esta clase utiliza el patrón *singleton* debido a que solo va a haber una instancia en toda la aplicación y facilita la comunicación con el resto de las clases que requieren de comunicarse con ella.

Pasando a la lógica de esta clase, destacamos los siguientes puntos:

- **Cola de eventos de ratón y teclado**. En primer lugar, tenemos una estructura de datos de tipo «cola» que almacena los objetos que contienen la información de los eventos de ratón y teclado que van llegando por el canal de datos. De este modo, continuamente con el método *Update()*, la clase comprueba si hay un evento nuevo en la cola para procesarlo. En caso de que lo haya, lo saca de la cola y en función del tipo de evento (ratón o teclado), llama al método manejador correspondiente pasando como parámetro el evento extraído.

Listado 5.1: Método QueueHandler() responsable de la gestión de la cola de eventos

```
private IEnumerator QueueHandler()
    {
        while (true)
        {
            if (eventQ.Count > 0)
            {
                if (eventQ.Peek() is HandInteractionData)
                    HandInteractionDataHandler
                    ((HandInteractionData)eventQ.Dequeue());
                else if (eventQ.Peek() is string)
                {
                    KeyHandler((string)eventQ.Dequeue());
                }
            }
            yield return null;
        }
    }
```

- **Manejador de eventos de teclado**. Es responsable del procesamiento de los eventos de teclado extraídos de la cola de eventos. Los eventos de teclado no requieren de mucho procesamiento; únicamente se llama al método de la librería de Windows «Forms» *SendWait* el cual directamente convierte el *string* recibido en pulsaciones de teclado. Cabe destacar que este *string* ha sido previamente tratado para adecuarse al formato de la librería.

Listado 5.2: Método KeyHandler() responsable de la gestión de los eventos de teclado

```
private void KeyHandler(string simpleKey)
    {
        SendKeys.SendWait(simpleKey);
    }
```

- **Manejador de eventos de ratón**. Se encarga de procesar los eventos de ratón extraídos de la cola de eventos. En este caso los eventos de ratón sí requieren de un mayor procesamiento debido a la variedad de tipos de interacción que pueden ocurrir con un ratón. En este proyecto hemos contemplado los siguientes tipos de interacción de "gesto de mano/ratón":
  - Click izquierdo.
  - Click derecho.
  - Arrastrar, es decir, mantener pulsado el click izquierdo y mover el ratón opcionalmente.
  - *scroll*, con la rueda del ratón.

  En función del tipo de interacción de ratón, se llama al método manejador correspondiente y se realizan las operaciones necesarias para generar la acción.

  Para procesar estos eventos y ejecutarlos se ha utilizado la clase *MouseEventManager*, la cual utiliza métodos propios de la librería «user32.dll» de Windows para ejecutar las acciones de ratón.

  El proceso que se seguiría, por ejemplo, para realizar un click izquierdo, sería el siguiente:

  1. Obtenemos la posición en pixeles donde se tiene que posicionar el puntero del ratón a partir del punto que se haya tocado del holograma en la aplicación de HoloLens 2.
  2. Posicionamos el puntero del ratón en esas coordenadas.
  3. Generamos un evento de presionar el botón de click izquierdo del ratón.
  4. Generamos un evento de levantar el botón de click izquierdo del ratón.

Listado 5.3: Método LeftClick() responsable de la función de click izquierdo

```
private void LeftClick(HandInteractionData ↩
    ↪ handInteractionData)
    {
    mouseEventManager.SetCursorPosition
    (GetScreenPoint(handInteractionData));
    mouseEventManager.MouseEvent
    (MouseEventManager.MouseEventFlags.LeftDown);
    mouseEventManager.MouseEvent
    (MouseEventManager.MouseEventFlags.LeftUp);
    }
```

2) Componentes relativos a la captura de pantalla

Otro de los bloques clave de esta aplicación es el que comprende los objetos que contienen la funcionalidad relativa a la captura de pantalla. Las clases que se encargan de esta tarea se encuentran agrupadas en el objeto *LocalMedia* de la escena. Sus nombres y su lógica es la siguiente:

Por un lado, tenemos la clase principal que es *ScreenVideoSource*. Esta clase contiene todos los métodos relativos a la captura del video. Implementa la clase abstracta de la biblioteca de WebRTC para Unity *CustomVideoSource* que permite el envío directo de *frames* en formato «ARGB32» a los componentes de WebRTC.

Cabe destacar que se ha utilizado la biblioteca de vínculos dinámicos «System.Drawing» de Windows para la captura de pantalla.

La lógica de la clase es la siguiente:

1. Primero se inicia un primer fotograma o *frame* con atributos asignados por defecto que será el que iremos actualizando con el vídeo en cuestión.
2. Iniciamos el hilo encargado de actualizar el *frame*, que realiza las siguientes acciones.
    a) Descubre la ventana de la aplicación con la ayuda de la clase *WindowDiscover* (se explicará posteriormente) y dibuja el icono del puntero.
    b) Crea un *bitmap* y lo bloquea en la memoria del sistema.
    c) Obtiene la dirección de memoria de los datos del primer pixel del mapa de bits y copia toda la información a un vector de tipo «byte» listo para ser consumido cuando sea necesaria una nueva imagen de la pantalla.

En segundo lugar, la otra clase relativa a la captura de pantalla es *WindowDiscover*. Esta clase tiene como función principal encontrar el manejador de la ventana de la aplicación que queremos capturar.

Para ello, utiliza varias estrategias; si la primera no ha conseguido encontrar el manejador, pasa a la siguiente. Las diferentes formas que se han implementado han sido:

A. Búsqueda del manejador por el proceso de la aplicación.
B. Búsqueda del manejador a partir del nombre de la ventana de la aplicación.
C. Selección de un manejador aleatorio entre todos los activos.

Esta clase utiliza la librería de Windows «User32.DLL» para obtener el nombre de una ventana, saber si una ventana es visible u obtener el identificador del proceso que creó una ventana a partir de su manejador.

La lógica de la clase es la siguiente:

1. **Intento de búsqueda del manejador por el proceso de la aplicación**
    1 Obtenemos el identificador del proceso de la aplicación a partir de su nombre.
    2 Obtenemos una lista con los manejadores de todas las ventanas abiertas y los identificadores asociados de los procesos que las iniciaron.
    3 Recorremos esta lista para encontrar el manejador cuyo identificador coincida con el del proceso de la aplicación.
    4 Una vez encontrado el manejador de la ventana, se devuelve y se envía a la clase *ScreenVideoSource* (comentada anteriormente).
2. **Intento de búsqueda del manejador por el proceso de la aplicación**
    1 Obtenemos una lista con los manejadores de todas las ventanas abiertas y los nombres de estas ventanas
    2 Recorremos esta lista para encontrar el manejador cuyo nombre asociado coincida con el nombre de la ventana de la aplicación que queramos capturar.
    3 Una vez encontrado el manejador de la ventana, se devuelve y se envía a la clase *ScreenVideoSource*.
3. **Selección de un manejador aleatorio entre todos los activos**
    1 Obtenemos una lista con los manejadores de todas las ventanas abiertas y los nombres de estas ventanas.
    2 Se selecciona una tupla de forma aleatoria, obtenemos el manejador y se envía a la clase *ScreenVideoSource*.

3) Componentes relativos a la gestión de los mensajes y del canal de datos

Por últimos tenemos el bloque que engloba los objetos que contienen la funcionalidad relativa al envío de mensajes y la gestión del canal de datos.

En este bloque destacaremos las siguientes clases:

- *ConnectionManager*
- *DesktopDataChannelStreamingManager*
- *DssServerRunner*

La clase principal que gestiona el establecimiento de la conexión con la aplicación de *HoloLens 2* y la apertura del canal de datos utilizando para todo ello WebRTC, es la clase *ConnectionManager*, que se encuentra en el *GameObject* de la escena de nombre homónimo.

Se trata de una clase que utiliza el patrón *singleton* que encapsula y simplifica el funcionamiento de MixedReality WebRTC permitiendo su uso de forma sencilla en cualquier aplicación. Las clases que maneja esta clase principal *ConnectionManager* y que contienen la lógica de MixedReality WebRTC se encuentran en la escena en el *GameObject* «WebRTC».

Es necesario que en la escena donde se use este *script* haya también presente un objeto del tipo «PeerConnection» para el establecimiento de la conexión con el otro dispositivo (las *HoloLens 2* en este caso).

A continuación se exponen algunos de los métodos y eventos principales de esta clase:

- ***Start()***: Busca el componente «PeerConnection» de WebRTC que haya en la escena y se subscribe a diferentes *listeners*. Entre ellos cabe destacar el *listener* que avisa cuando una conexión esta lista para ser utilizada.

- ***OnPeerConnectionInitialized()***: Evento que se lanza cuando la conexion está lista para ser utilizada. A su vez se suscribe a otros *listener*, como:
  - *IceStateChanged*, el cual avisa cuando cambia el estado de la conexión, algo necesario para saber cuando empieza y termina una llamada.
  - *DataChannelAdded*, el cual avisa cuando se añade un nuevo canal de datos.
- ***IceStateChangedDelegate()***: Como se ha explicado arriba, este es el método al que se llama cuando cambia el estado de la conexión. En función del nuevo estado se realizarán unas funciones u otras:
  - Estado *new*. El agente ICE está esperando candidatos remotos para evaluarlos. Es el estado inicial.
  - Estado *checking*. El agente ICE ha recibido uno o más candidatos remotos y está intentando encontrar pares compatibles entre los candidatos locales.
  - Estado *connected*. Se ha encontrado al menos un par compatible de candidatos y se ha establecido la conexión.
  - Estado *completed*. El agente ICE ha logrado encontrar una conexión para todos los componentes.
  - Estado *failed*. No se han encontrado candidatos ICE compatibles. Es imposible por tanto iniciar la conexión y se llama al método de finalizar llamada.
  - Estado *disconnected*. Significa que el *peer* ha colgado llamada o se ha caído, y por lo tanto se llama al método de finalizar llamada.
  - Estado *closed*. La conexión se ha cerrado completamente y se llama al método de finalizar llamada.
- ***DataChannelAddedEvent()***: Evento de vital importancia pues posibilita el inicio del flujo de acciones al añadirse un canal de datos. Este flujo de acciones consistirá básicamente en la identificación del canal de datos añadido, guardándolo en una variable para trabajar con él y suscribirse al evento «MessageReceived» el cual se llama cuando se recibe un mensaje a través de este canal de datos. Los mensajes recibidos por este canal consistirán en los objetos de interacción enviados desde la aplicación de las *HoloLens 2* para generar los eventos de teclado y ratón en el ordenador.

Pasando a la siguiente clase destacable, *DesktopDataChannelStreamingManager*, es la clase responsable de gestionar el canal de datos abierto con la aplicación de *HoloLens 2* en el lado del ordenador.

Dentro de esta clase destaca el método de suscripción a los eventos de nuevos mensajes «OnDataChannelMessage», que se lanzará cada vez que llegue un mensaje nuevo (un objeto de interacción).

Listado 5.4: Método SubscribeToMessageReceivedEvent()

```
1  public void SubscribeToMessageReceivedEvent(DataChannel ↩
       ↪ dataChannel)
2      {
3          Debug.Log("Data␣channel␣added");
4          JaltestDataChannel = dataChannel;
5          JaltestDataChannel.MessageReceived += OnDataChannelMessage;
6          Debug.Log("Subscribed␣to␣Data␣channel␣message␣events");
7      }
```

Estos mensajes llegan en formato de array de bytes, por lo que tienen que ser transformados a objetos mediante otro método de esta clase llamado *ByteArrayToObject()*, el cual utiliza la clase *BinaryFormatter* que sirve para serializar o deserializar un objeto en formato binario.

Listado 5.5: Método ByteArrayToObject() para deserializar objetos

```
1   public static object ByteArrayToObject(byte[] arrBytes)
2     {
3         using (var memStream = new MemoryStream())
4         {
5             var binForm = new BinaryFormatter();
6             memStream.Write(arrBytes, 0, arrBytes.Length);
7             memStream.Seek(0, SeekOrigin.Begin);
8             var obj = binForm.Deserialize(memStream);
9             return obj;
10        }
11    }
```

Por último, una vez deserializado el objeto, se envía a la clase *DesktopEventsTrigger* para que gestione la interacción en función de su tipo.

La última clase relativa a la parte de conexión de la aplicación de escritorio es *DssServerRunner*. Esta clase es la encargada de arrancar un servidor local en el puerto 3000 para permitir el establecimiento de la conexión entre ambas aplicaciones.

Listado 5.6: Método runDss() para iniciar el servidor DSS

```
1    private IEnumerator runDss()
2     {
3         dssProcess = new Process();
4         dssProcess.StartInfo.WindowStyle = ↩
              ↪ ProcessWindowStyle.Hidden;
5         dssProcess.StartInfo.FileName = filePath;
6         dssProcess.StartInfo.Arguments = "3000␣-␣v";
7         dssProcess.Start();
8
9         yield return null;
10    }
```

El servidor se arranca de forma invisible al usuario; la ventana de la aplicación no es visible al usuario, aunque el proceso puede verse en el administrador de tareas.Además, cuando la aplicación se cierra, esta clase es la encargada de terminar con el proceso del servidor.

5.2.2. Aplicación de *HoloLens 2*

A continuación se expone un diagrama de clases simplificado que recoge los subsistemas expuestos previamente de la aplicación de *HoloLens 2* con sus clases, atributos, métodos y relaciones más significativas.

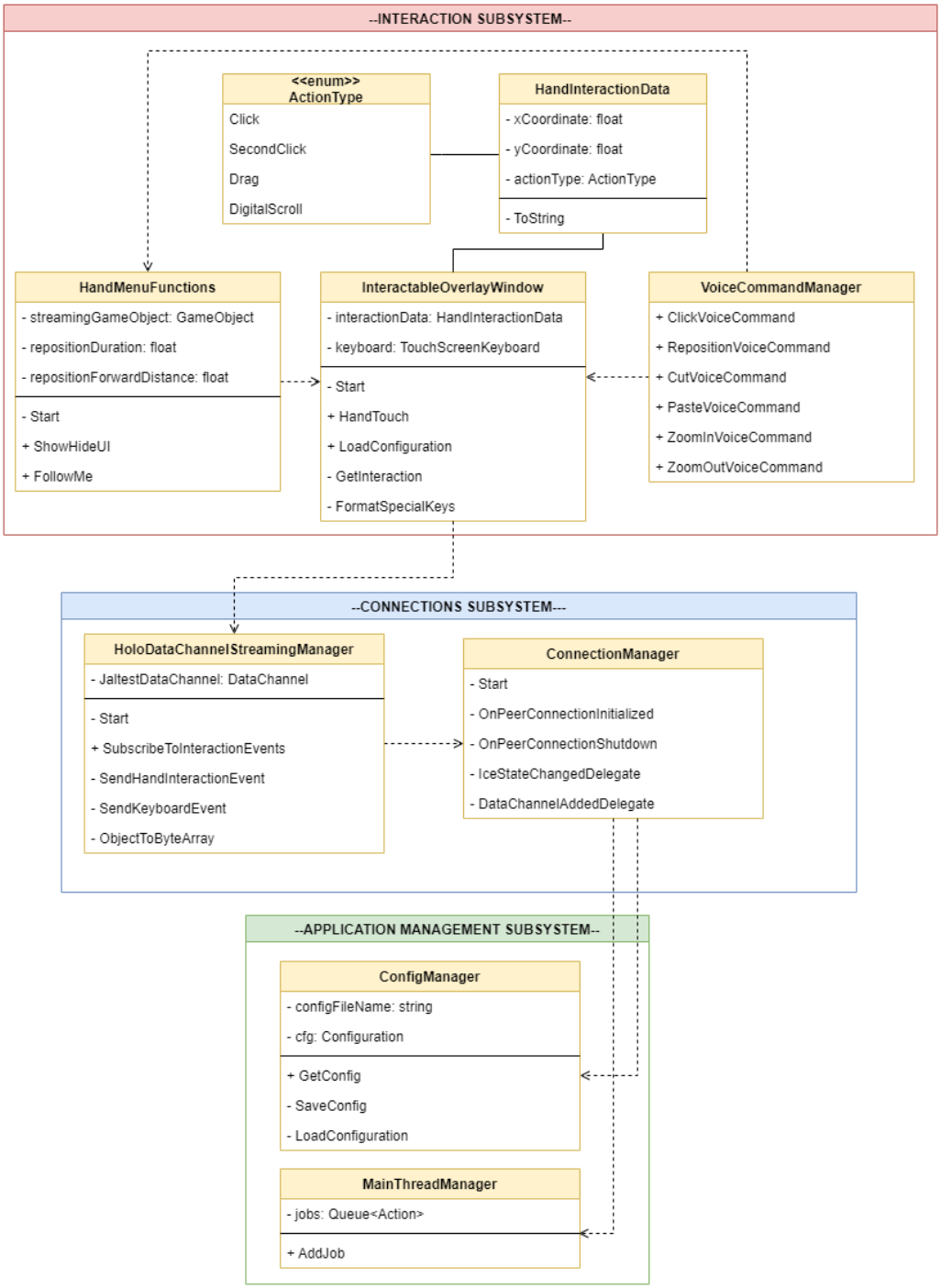

Figura 5.6: Diagrama de clases de la aplicación de *HoloLens 2* del módulo 1)

Los objetos que componen la escena son los siguientes:

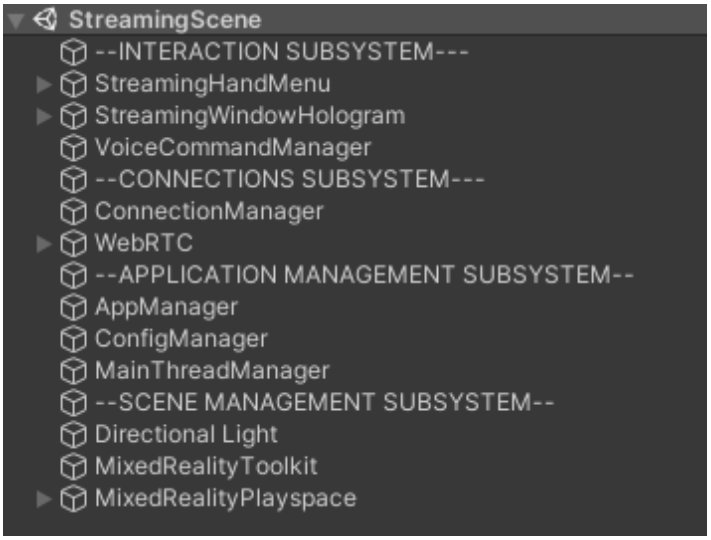

Figura 5.7: Componentes de la escena de la aplicación de *HoloLens 2* del módulo 1 en Unity usando GameObjects vacíos como separadores

En el lado de la aplicación de *HoloLens 2* existen objetos y clases idénticas al lado de la aplicación de escritorio, como las que se encargan del establecimiento de la conexión mediante Mixed Reality WebRTC.

Como en la aplicación de escritorio, se ha seguido la misma lógica a la hora de organizar los componentes de la escena, es decir, por responsabilidades.

Los tres grupos de objetos más importantes de esta escena y por tanto de esta aplicación son:

1) Componentes relativos a la interacción con el holograma de la ventana de *streaming*.
2) Componentes relativos a la gestión de los objetos de interacción.
3) Componentes relativos a la gestión de los comandos de voz.

Componentes relativos a la interacción con el holograma de la ventana de *streaming*.

La explicación de esta sección se va a realizar atendiendo a los tipos de interacción que se pueden producir por parte del usuario:

A. **Interacción de tipo gestual**. En este grupo se engloban:
   - Interacción por contacto directo: click izquierdo, click derecho y arrastrar.
   - Interacción a distancia mediante *RayClick*: click izquierdo.

B. **Interacción con teclado holográfico**: generación de eventos de pulsaciones de teclado.

C. **Interacción por comandos de voz**: genera pulsaciones o combinaciones de pulsaciones de teclado.

D. **Interacción con botones de *scroll***: *scroll* hacia arriba y *scroll* hacia abajo.

Comenzando pues por la interacción de tipo gestual, esta puede ser por contacto directo o a distancia. Ambas se producen sobre el componente de la ventana de *streaming*. El responsable de controlar la lógica y los componentes de la ventana se llama «RemoteVideo», y dentro de él se encuentra un objeto hijo llamado «RemoteVideoPlayer» que contiene la clase *InteractibleOverlayWindow*. Esta clase es la encargada de gestionar las interacciones de este tipo y su posterior conversión en objetos de interacción.

A continuación se destaca la lógica y métodos más importantes de esta clase:

- ***HandTouch()***: se trata del método al que se llama cuando se recibe un aviso de interacción por contacto, es decir, un evento que avisa de que se ha producido un contacto de la mano del usuario con el holograma.

  A partir de este aviso, el método debe comprobar qué tipo de interacción se ha producido. Como vimos en la parte de la aplicación de ordenador, hemos contemplado tres posibles interacciones con las manos: click izquierdo, click derecho y arrastrar.

  Cabe mencionar que *scroll*, en un principio, se implementó como interacción gestual, pero debido a problemas de usabilidad, se decidió implementarlo a partir de un par de botones laterales que envían un "impulso" de *scroll* (se detallará su funcionamiento posteriormente).

  Por lo tanto, el método debe comprobar si la interacción se trata de un click izquierdo, derecho o arrastrar.

  En este punto hubo que idear formas para diferenciar entre estas interacciones, pues de manera nativa MRTK no contiene soluciones para esta problemática.

  Entonces, a partir de la posición del holograma de los dedos del usuario se configuraron varios métodos que comprueban en que posición se encuentran para saber que tipo de interacción se está realizando. De este modo tenemos las siguientes posiciones:

  - Para click izquierdo se estableció que el dedo índice debía estar estirado (se toca con este dedo) y el dedo pulgar también debía estar estirado formando un ángulo de 90º.
  - Para click derecho se determinó que el dedo índice debía estar estirado y el dedo pulgar debía estar recogido agarrando los otros dedos de la mano por debajo.
  - Para arrastrar se estableció que el contacto con la ventana se debía de realizar haciendo un gesto de pellizcar con el dedo índice y pulgar (*pinch* en inglés) y mantener el contacto hacia la posición donde se desea que se produzca el desplazamiento.

  El tipo de interacción se guarda como una variable de tipo «enum» creada *adhoc*.

Listado 5.7: Clase de tipo enum *ActionType*

```
 public enum ActionType
{
    Click,
    SecondClick,
    MouseClick,
    Drag,
    DragEnd,
    Scroll,
    Digital scroll
}
```

Continuando con la explicación, una vez se ha identificado el tipo de interacción se pasa a obtener el objeto de interacción, pero antes de continuar, es conveniente introducir la clase que encapsula la lógica de estos objetos de interacción: *HandInteractionData*. Se trata de una clase que recoge la información relativa a la interacción producida con la ventana de *streaming* para que pueda ser fácilmente encapsulada y enviada a la aplicación de escritorio.

Listado 5.8: Clase *HandInteractionData* para encapsular la lógica de los objetos de interacción gestual

```
 public class HandInteractionData
{
    private float xCoordinate;
    private float yCoordinate;
    private ActionType actionType;

    public HandInteractionData(float xCoordinate, float yCoordinate, ActionType actionType)
    {
        this.xCoordinate = xCoordinate;
        this.yCoordinate = yCoordinate;
        this.actionType = actionType;

    }

    public float XCoordinate { get => xCoordinate; set => xCoordinate = value; }
    public float YCoordinate { get => yCoordinate; set => yCoordinate = value; }
    public ActionType ActionType { get => actionType; set => actionType = value; }

    public override string ToString()
    {

        return this.xCoordinate + "␣" + this.yCoordinate + "␣" + this.actionType;

    }

}
```

De este modo, un objeto de interacción de mano está compuesto por la coordenada X e Y relativa donde ha tocado el usuario la ventana y el tipo de enumeración «actionType» que identifica el tipo de acción de la interacción.

Continuando con el proceso de obtención del objeto de interacción de mano, se llama al método *GetInteraction()* pasándole como argumento el tipo de interacción. Entonces deberá comprobar:

a. Si se está produciendo actualmente una interacción de tipo "drag/arrastrar", para continuar enviando las posiciones y manteniendo activada la corrutina encargada de comprobar que el contacto con la pantalla se sigue produciendo.

b. Si no se ha estado produciendo una interacción de «Drag» pero se inicia en este momento, entonces deberá arrancar la corrutina que comprueba el contacto de la mano con la pantalla y el envío continuo de las posiciones a través de los objetos de interacción. En el momento en que se deje de producir el contacto se enviará una acción de tipo «DragEnd» para levantar el click en el ordenador.

c. Si es de tipo click izquierdo o click derecho simplemente deberá de recoger la posición en coordenadas, crear el objeto y devolverlo al método *HandTouch()* desde el que se ha llamado a este método.

Una vez que se ha obtenido el objeto de interacción, se genera un nuevo evento de tipo «OnScreenInteraction» pasando como argumento el objeto de interacción. A este canal están suscrita la clase *HoloDatachannelStreamingManager*, que se explicará posteriormente.

Por último, queda hablar sobre la interacción gestual a distancia, la cual no está basada en el contacto directo sino que utiliza el *RayClick*, que es un rayo a distancia que sale del dedo índice del usuario y colisiona con el primer objeto que encuentre en su trayectoria. Si el usuario realiza un gesto de pellizcar teniendo este rayo activado, se producirá un click en la posición en que este el rayo del mismo modo que si se hubiera realizado mediante contacto directo.

Este tipo de interacción se controla a través del método *RayClick()* que es llamado cuando se produce un evento de tipo «OnClick» sobre la ventana de *streaming*. Simplemente genera un objeto de interacción de tipo «Click» con las coordenadas de la colisión del rayo sobre la ventana de *streaming*.

Una vez expuesta la parte de la interacción gestual, se procede a explicar la parte de la interacción con teclado holográfico.

La interacción mediante el teclado holográfico se produce usando el componente nativo de HoloLens 2 para este fin.

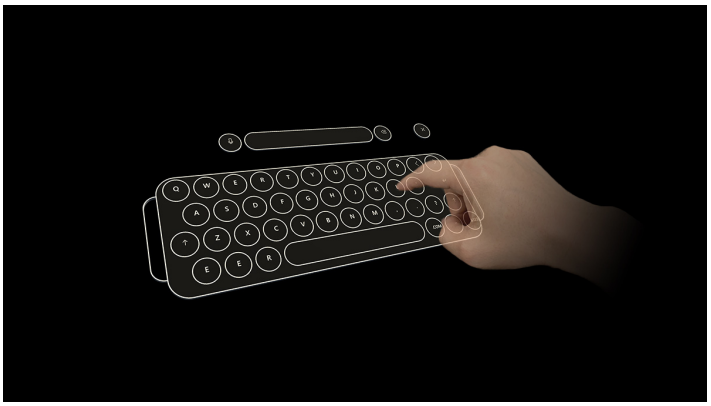

Figura 5.8: Teclado nativo del Mixed Reality ToolKit utilizando en la plataforma *Apholo*

La lógica de este tipo de interacción es la siguiente: cada vez que una tecla es pulsada se genera un evento de pulsación que es recogido. Este evento se transforma en un string que almacena el carácter de la tecla pulsada.

Las teclas normales no necesitan una conversión explícita («KeyCode.A» se convierte a A directamente, por ejemplo), pero hay teclas especiales que tienen su formato definido por Windows para la generación de eventos[1] (por ejemplo, la tecla *Escape* sería "{Esc}"). Para la conversión de estas teclas especiales se utiliza un diccionario que convierte los objetos de tipo «KeyCode» en *strings* con el formato adecuado.

[1] https://docs.microsoft.com/en-us/dotnet/api/system.windows.forms.sendkeys.send?view=windowsdesktop-6.0

Listado 5.9: Diccionario para convertir las teclas especiales en string comprensibles por la biblioteca de Windows

```
  private Dictionary<KeyCode, string> keycodeToString = new ↩
      ↪ Dictionary<KeyCode, string>()
{
        {KeyCode.Backspace, "{BACKSPACE}" },
        {KeyCode.Delete, "{DELETE}" },
        {KeyCode.Tab, "{TAB}" },
        {KeyCode.DownArrow, "{DOWN}" },
        {KeyCode.UpArrow, "{UP}" },
        {KeyCode.RightArrow, "{RIGHT}" },
        {KeyCode.LeftArrow, "{LEFT}" },
        {KeyCode.Escape, "{ESC}" },
        {KeyCode.KeypadEnter, "{ENTER}" },
        {KeyCode.Return, "{ENTER}" },
        {KeyCode.Help, "{HELP}" },
        {KeyCode.Home, "{HOME}" },
        {KeyCode.Insert, "{INSERT}" },
        {KeyCode.F1, "{F1}" },
        {KeyCode.F2, "{F2}" },
        {KeyCode.F3, "{F3}" },
        {KeyCode.F4, "{F4}" },
        {KeyCode.F5, "{F5}" },
        {KeyCode.F6, "{F6}" },
        {KeyCode.F7, "{F7}" },
        {KeyCode.F8, "{F8}" },
        {KeyCode.F9, "{F9}" },
        {KeyCode.F10, "{F10}" },
        {KeyCode.F11, "{F11}" },
        {KeyCode.F12, "{F12}" },
        {KeyCode.F13, "{F13}" },
        {KeyCode.F14, "{F14}" },
        {KeyCode.F15, "{F15}" },
        {KeyCode.KeypadPlus, "{ADD}" },
        {KeyCode.KeypadMinus, "{SUBTRACT}" },
        {KeyCode.KeypadMultiply, "{MULTIPLY}" },
        {KeyCode.KeypadDivide, "{DIVIDE}" }
};
```

Para la interacción con teclado no se ha requerido de la creación de una clase para encapsular su información ya que lo único que se necesita es un *string* que almacena el carácter o caracteres que van a ser generados. El método de la librería *dll* de Windows en el ordenador de escritorio es capaz de convertir cada uno de estos caracteres en las pulsaciones de teclas asociadas. También es capaz de generar eventos de combinaciones de teclas, como “Ctrl+C”, siguiendo un formato específico en el string, el cual también se contempla en esta aplicación.

Respecto a la interacción por comandos de voz, esta se aprovecha de la lógica para enviar eventos de teclado, ya que los comandos de voz que se han implementado están asociados a combinaciones de pulsaciones de teclas para realizar funciones tales como acercar (*zoom in*), alejar (*zoom out*), cortar texto, etc. Se ampliará la explicación de este punto en el apartado sobre los componentes relativos a la gestión de comandos de voz.

Por último, tenemos la interacción con botones de *scroll*, la cual simplifica enormemente la implementación de dicha función. El funcionamiento consiste en la creación de un objeto de interacción de tipo «Scroll» con coordenadas Y positivas o negativas en función de si el botón pulsado es *Scroll Up* o *Scroll Down*.

Listado 5.10: Método Send*scroll*Data() para enviar un objeto de interacción de tipo *scroll*

```
private void Send\emph{scroll}Data(float yCoordinate)
  {
      HandInteractionData \emph{scroll}Data = new ↩
          ↪ HandInteractionData(0, yCoordinate, ↩
          ↪ ActionType.Digital\emph{scroll});
      Debug.Log("Sending␣\emph{scroll}␣data...");
      overlay.SendRandomScreenInteraction(\emph{scroll}Data);
  }
```

Este objeto se envía a través del canal dedicado a los objetos de interacción con la ventana de *streaming*, «OnScreenInteraction», para que sea gestionado por el objeto encargado de ello (*HoloDataChannelStreamingManager*).

Componentes relativos a la gestión de los objetos de interacción

La gestión de los objetos de interacción es otra de las partes vitales de esta aplicación. Esta gestión consiste básicamente en:

1. La recepción de los diferentes objetos de interacción desde las clases que los han construido.
2. La conversión de dichos objetos en array de bytes.
3. El envío de estos array de bytes a la aplicación de ordenador a través del canal de datos de Mixed Reality WebRTC.
4. La apertura del canal de datos para poder enviar estos objetos.

La clase encargada de ello es *HoloDataChannelStreamingManager*, que se encuentra en el objeto de la escena «ConnectionManager».

Los métodos más importantes de esta clase son:

- ***SubscribeToInteractionEvents():*** este método se lanza cuando se recibe el aviso de que el canal de datos ha sido añadido. En ese momento se añaden *listeners* a los eventos de interacción, tanto gestual, que enviaría un objeto de tipo «HandInteractionData», como de teclado, que enviaría un *string*.
- ***SendHandInteractionEvent():*** convierte el objeto de interacción gestual en un array de bytes utilizando el método *ObjectToByteArray()* de esta clase y posteriormente lo envía por el canal de datos si este ha sido abierto.
- ***SendKeyBoardEvent():*** convierte el *string* que almacena las pulsaciones de teclado en un array de bytes utilizando el método *ObjectToByteArray()* de esta clase y posteriormente lo envía por el canal de datos si este ha sido abierto.

Componentes relativos a la gestión de los comandos de voz.

Finalmente el último bloque destacable de esta aplicación es el relativo a los componentes encargados de gestionar los comandos de voz.

Las clases responsables de ello se encuentra en el objeto de la escena «VoiceCommandManager»: la clase *SpeechInputHandler* es nativa de MRTK y permite configurar los comandos de voz que hayamos añadido en el componente de configuración general de la escena de MRTK. Los comandos deben ser especificados tal y como suenan al pronunciarse. Por ello, y tras la realización de pruebas, se comprobó que el comando de *Zoom In* (acercar) era mejor identificado por las gafas si se especificaba textualmente como «Zumin». El resto si se han escrito de forma similar a como se dirían en castellano o inglés en el caso de *Zoom Out*.

En esta misma clase además se gestiona el evento al que se llama cuando es identificado el comando de voz en cuestión:

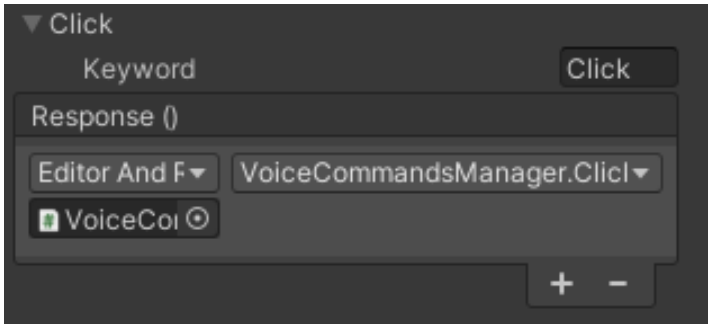

Figura 5.9: Gestión de los comandos de voz a través del editor de Unity

La clase responsable de las funciones asociadas a los comandos de voz es *VoiceCommandManager*.

A continuación se exponen los comandos de voz que se han implementado, su dicción y la acción que producen:

Tabla 5.1: Tabla con los comandos de voz implementados

Nombre	Pronunciación	Efecto
Control + X	Control Equis	Envía el string "^X" que inicia en el ordenador el evento de pulsación de las teclas "Control" y "X" simultáneamente. Utilizado para cortar una cadena de texto seleccionada.
Control + C	Control Ce	Envía el string "^C" que inicia en el ordenador el evento de pulsación de las teclas "Control" y "C" simultáneamente. Utilizado para copiar una cadena de texto seleccionada.
Control + V	Control Uve	Envía el string "^V" que inicia en el ordenador el evento de pulsación de las teclas "Control" y "V" simultáneamente. Utilizado para pegar una cadena de texto del portapapeles.
RayClick	Click	Llama al método RayClick de la clase InteractibleOverlayWindow para realizar un click a distancia.
Acercar	Zumin	Envía el string "^{+}" que inicia en el ordenador el evento de pulsación de las teclas "Control" y "+" simultáneamente. Utilizado para ejecutar el comando ampliar de determinadas aplicaciones.
Alejar	Zoom Out	Envía el string "^{-}" que inicia en el ordenador el evento de pulsación de las teclas "Control" y "-" simultáneamente. Utilizado para ejecutar el comando alejar de determinadas aplicaciones.
Reposicionar	Reposicionar	Llama el método RepositionUI de la clase HandMenuFunctions para llevar la ventana de streaming al frente del usuario.

El siguiente fragmento de código muestra la gestión de los diferentes comandos por la clase *CommandVoiceManager*.

Listado 5.11: Métodos para gestionar los comandos de voz de la clase *VoiceCommandsManager*

```
public void ClickVoiceCommand()
 {
     if (OnVoiceCommandInteraction != null)
         interactibleOverlayWindow.RayClick();

 }
 public void RepositionVoiceCommand()
 {
     if (OnVoiceCommandInteraction != null)
         handMenuFunctions.RepositionUI();
 }

 public void CutVoiceCommand() //Send ctrl + X
 {
     if (OnVoiceCommandInteraction != null)
         OnVoiceCommandInteraction("^X");
 }

 public void PasteVoiceCommand()  // Send ctrl + V
 {
     if (OnVoiceCommandInteraction != null)
         OnVoiceCommandInteraction("^V");
 }

 public void ZoomInVoiceCommand()  // Send ctrl + «+»
 {
     if (OnVoiceCommandInteraction != null)
         OnVoiceCommandInteraction("^{+}");
 }

 public void ZoomOutVoiceCommand()  // Send ctrl + «-»
 {
     if (OnVoiceCommandInteraction != null)
         OnVoiceCommandInteraction("^{-}");
 }
```

Como se puede apreciar no todos los comandos siguen el mismo flujo de procesamiento una vez capturados. En función de su naturaleza, unos se gestionarán como si fueran comandos gestuales, otros llamarán a funciones del menú mano y otros se gestionarán como si fueran eventos de teclado a través del canal *OnVoiceCommandInteraction()*.

5.3. MÓDULO 2. APLICACIÓN DE SOPORTE REMOTO POR VIDEOLLAMADA

Como en el apartado anterior, se comenzará el epígrafe con la exposición de un diagrama de subsistemas de este módulo 2. Posteriormente se pasará a explicar aquellos componentes que sean más importantes o complejos.

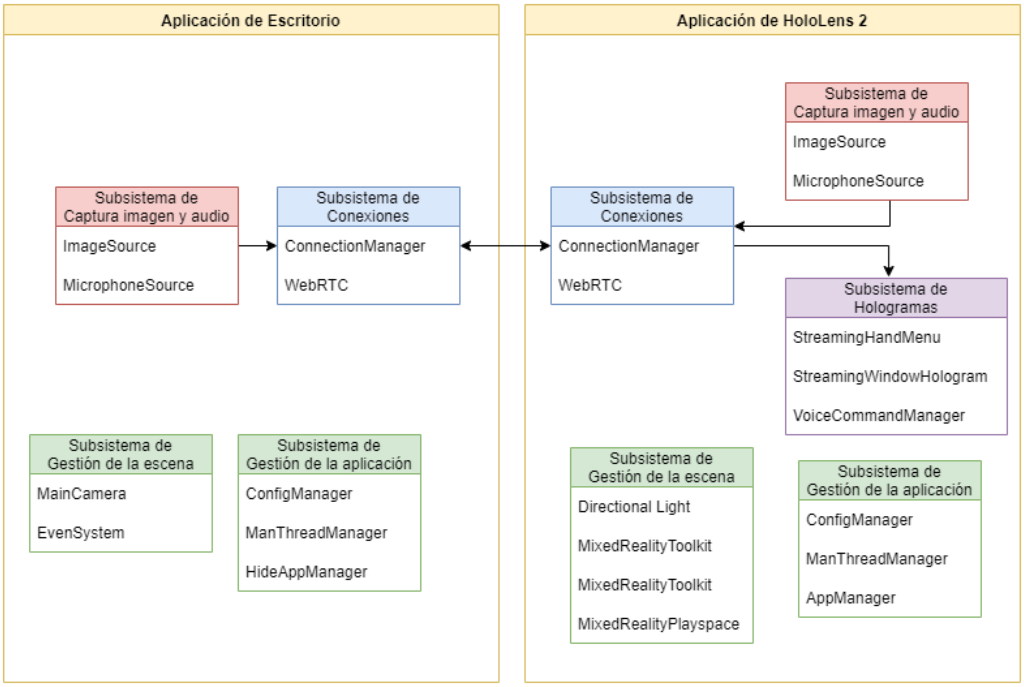

Figura 5.10: Subsistemas que componen el módulo 2 de soporte remoto por videollamada

El objetivo de este módulo es el de poder establecer una videollamada entre un técnico de soporte y un trabajador del sector industrial. La funcionalidad de cada aplicación de este módulo es la siguiente:

- **Aplicación de escritorio**: recoge la imagen y el sonido capturado por una *webcam* y los envía hacia la aplicación de *HoloLens 2* para que la imagen sea mostrada en el componente de ventana holográfica y el audio sea reproducido por los altavoces del dispositivo de realidad mixta.
- **Aplicación de *HoloLens 2***: de forma análoga, recogerá la imagen y el sonido capturado por la cámara y los micrófonos de las *HoloLens 2* y los enviará hacia la aplicación de escritorio para que la imagen sea mostrada en dicha aplicación y el audio reproducido por el ordenador.

Este módulo, como se comentará más detalladamente a continuación, comparte la mayor parte de los componentes con el módulo anterior de *streaming*, ya que lo que se pretende es en buena medida el envío de una imagen hacia la otra aplicación. Sin embargo se pueden observar algunas diferencias con los subsistemas del anterior módulo, como la adición de una fuente de audio («MicrophoneSource») ya que ahora también se transmite el audio recogido por un micrófono, o el subsistema de Hologramas, que en este caso no tiene funcionalidad de interacción sino de solo mostrar la imagen que llega por el canal de datos.

5.3.1. Explicación de la aplicación

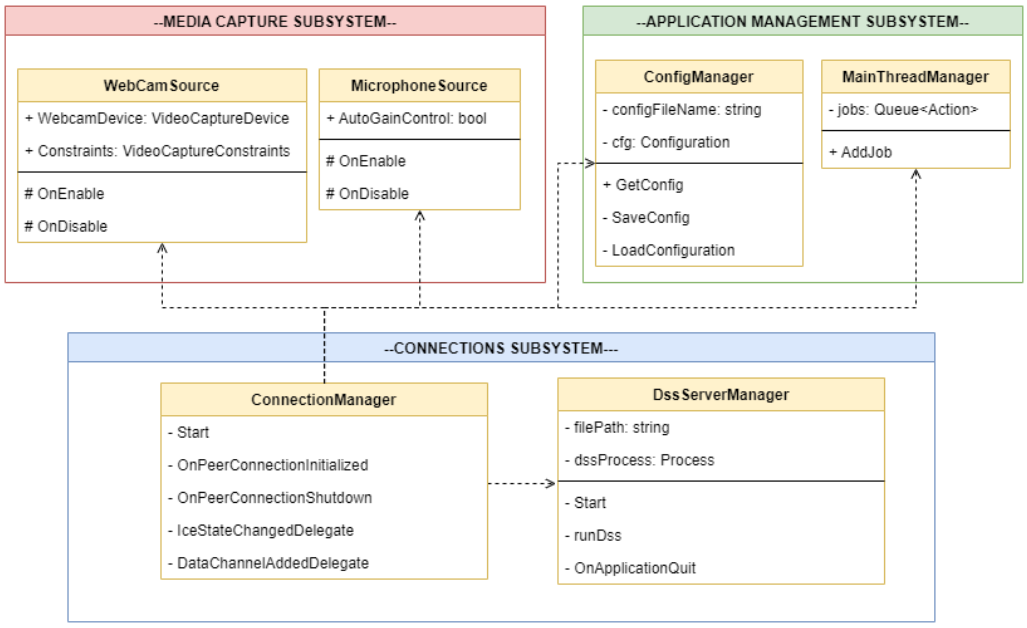

Figura 5.11: Diagrama de clases de la aplicación de escritorio del módulo 2

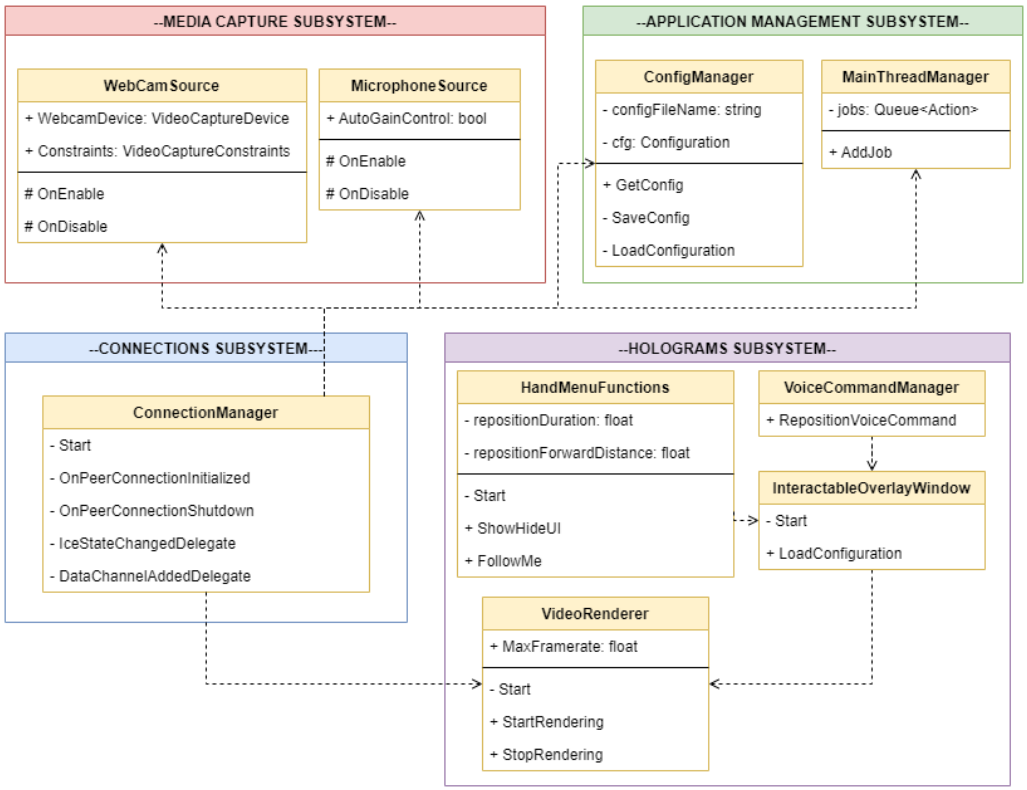

Figura 5.12: Diagrama de clases de la aplicación de HoloLens 2 del módulo 2

Para no extender demasiado la exposición de esta parte de forma innecesaria, se comentarán las principales diferencias respecto al módulo anterior, ya que este módulo de videoconferencia comparte la mayoría de las características con el módulo de *streaming*: interfaz de ambas aplicaciones, forma de iniciar la llamada, canales de datos, etc.

De este modo, las diferencias principales con respecto al módulo de *streaming* son:

- La transmisión de imágenes ya no es unidireccional (ordenador → *HoloLens 2*) sino bidireccional, ya que el usuario de las gafas debe poder ver a su técnico de soporte en la ventana-holograma, y el técnico de soporte que está en el PC debe ver la imagen recogida por las gafas para brindar la ayuda adecuada. Por lo tanto, ambas aplicaciones comparten los mismos componentes tanto para capturar la imagen como para renderizarla.
- Como se ha introducido en el punto anterior, ya no se captura la pantalla de una aplicación sino la imagen recogida por una *webcam* (en el caso de la aplicación del ordenador) o la imagen recogida por la cámara de las *HoloLens 2* (en el caso de la aplicación de las *HoloLens 2*). Esto se consigue cambiando la fuente de la imagen, sustituyendo la clase *ScreenVideoSource* por *WebCamSource*, ambas de la biblioteca de WebRTC para Unity, y actualizando los campos convenientes en el resto de las clases.
- Por último, toda la funcionalidad relativa a la generación de eventos de teclado y ratón y la captura de la interacción por parte del usuario con la ventana del holograma ha sido eliminada ya que en este módulo el propósito no es ese, sino establecer una videollamada entre el ordenador y las *HoloLens 2*.

5.4. MÓDULO 3: APLICACIÓN CON COMPONENTES NATIVOS DE HOLOLENS 2.

Primeramente, como en el apartado anterior, se van a mostrar una serie de diagramas para comprender la lógica general de esta aplicación y pasar posteriormente a una explicación más pormenorizada.

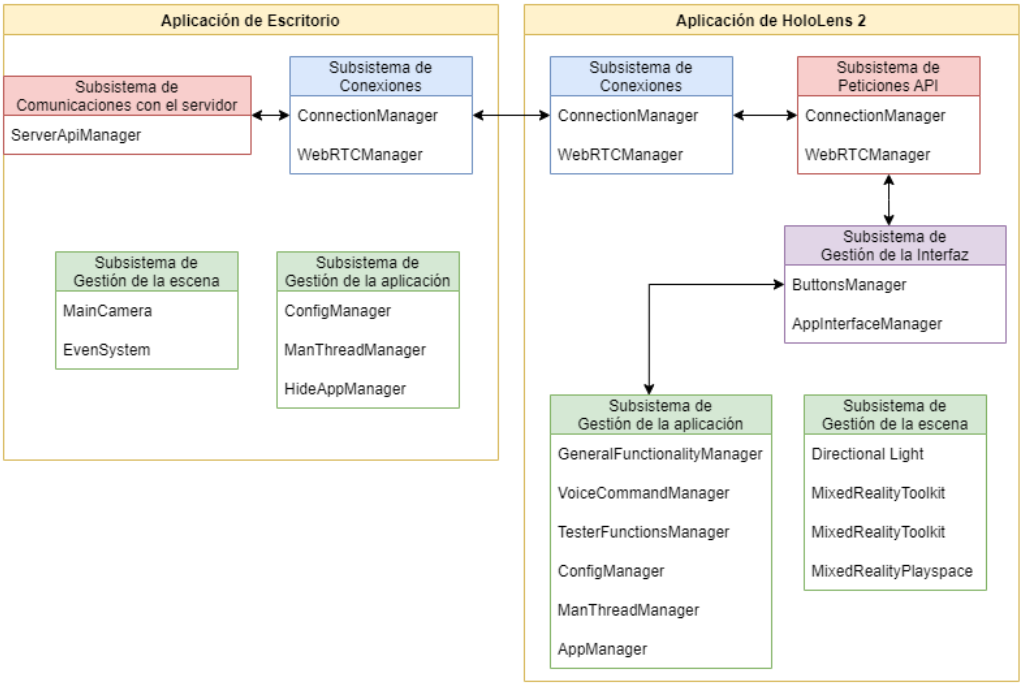

Figura 5.13: Subsistemas que componen el módulo 3

Los subsistemas principales de la aplicación de escritorio son:

- **Subsistema de comunicaciones con el servidor**: es el encargado de construir y gestionar el envío y recepción de peticiones HTTP al servidor web asociado a la aplicación industrial con la que se esté trabajando para la obtención de información relativa a los objetos de negocio que se necesitan para el funcionamiento de la aplicación de *HoloLens 2*.
- **Subsistema de conexiones**: es el responsable de establecer y gestionar la conexión con la aplicación de *HoloLens 2* para transmitir la respuesta recibida por el servidor web y recibir las demandas de peticiones HTTP generadas a partir de la interacción del usuario de las *HoloLens 2* con los menús de la aplicación realidad mixta.
- **Subsistema de gestión de la aplicación**: subsistema encargado de algunas funciones generales como la gestión de un archivo de configuración para guardar y cargar atributos de configuración, gestionar el hilo principal de la aplicación (es el único que puede realizar ciertas acciones), o hacer que la aplicación se ejecute en segundo plano.
- **Subsistema de gestión de la escena**: subsistema con objetos predeterminados y propios de Unity para la gestión de la escena.

Por otra parte, los subsistemas más importantes de la aplicación de *HoloLens 2* son:

- **Subsistema de peticiones API**: subsistema responsable de la generación y gestión de objetos de petición HTTP a partir de la interacción del usuario con los diferentes menús holográficos y también de la gestión de los objetos de respuesta de estas peticiones recibidos a través del canal que comunican la aplicación de escritorio con la de *HoloLens 2*.

- **Subsistema de gestión de la interfaz**: subsistema responsable de la construcción y gestión de la interfaz gráfica de la aplicación de *HoloLens 2*, incluyendo los menús, botones, su redimensionamiento, la generación de eventos a partir de la interacción del usuario que dan como resultado la creación de peticiones, etc.
- **Subsistema de conexiones**: componente análogo al de la aplicación de escritorio, con mismos objetos y funciones. Gestiona el canal por el que se transmitirán los objetos de peticiones y se recibe la respuesta recibida por el servidor en forma de objeto de negocio.
- **Subsistema de gestión de la aplicación**: Contiene funcionalidad general de la aplicación, como un componente gestor del idioma y los textos, un componente con funciones de *testing*, etc.
- **Subsistema de gestión de la escena**: comparte la funcionalidad con el mismo sistema de la aplicación de escritorio, pero incluye los componentes adicionales de MRTK para la gestión de proyectos de realidad mixta.

5.4.1. Aplicación de escritorio

A continuación se muestra un diagrama de clases simplificado que expone los subsistemas expuestos previamente con sus clases, atributos, métodos y relaciones más significativas de la aplicación de escritorio del módulo 3 de la platafora *Apholo*.

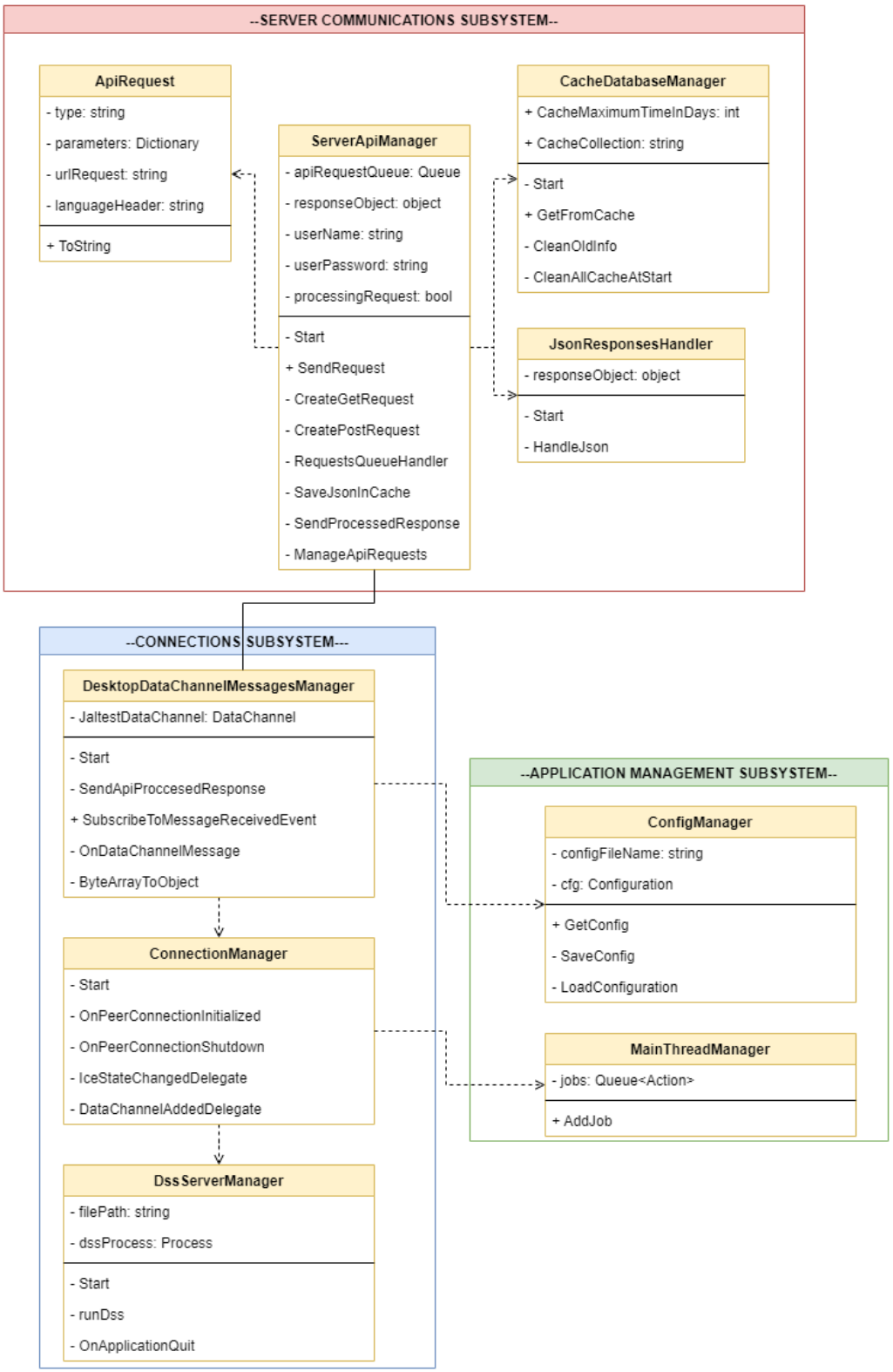

Figura 5.14: Diagrama de clases de la aplicación de escritorio del módulo 3

En el módulo 3 “Aplicación con componentes nativos de HoloLens 2”, se han reutilizado algunos componentes y estructuras de los módulos anteriores.

Las funciones de esta aplicación son:

A. Contener y encapsular todo lo posible la lógica del sistema en cuanto a clases y procesamiento de la información y en cuanto a comunicación con la API para realizar peticiones HTTP de la que se conseguirá la información necesaria que será mostrada en la aplicación de HoloLens 2.
B. Realizar el máximo volumen posible de cálculos y procesamiento de la información para mejorar el rendimiento de la aplicación para HoloLens 2.
C. Realizar peticiones HTTP a la API web del sistema en cuestión.
D. Iniciar la llamada para establecer la conexión con la aplicación de HoloLens 2.

Los objetos de la escena son los siguientes

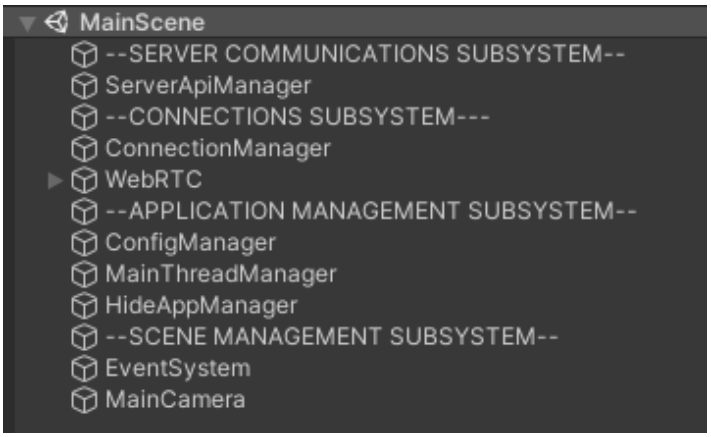

Figura 5.15: Componentes de la escena de la aplicación de escritorio del módulo 3

Como ya se ha comentado, y ahora se puede observar en la escena, hay muchos objetos que se repiten de las anteriores aplicaciones (todo lo relativo a la conexión y utilidades generales, por ejemplo). Estos puntos no se volverán a comentar en este apartado aunque sean importantes para el funcionamiento de la aplicación.

Por lo tanto se procede a destacar aquellos componentes novedosos e importantes para el funcionamiento de este módulo y de esta aplicación de ordenador en concreto:

1) Componentes relativos a la comunicación con la API del servidor web.
2) Componentes relativos al procesamiento de los JSON devueltos por el servidor web.

1) Componentes relativos a la comunicación con la API del servidor web

La comunicación con la API del servidor web es sin duda una de las partes más importantes de esta aplicación, ya que este módulo se ha diseñado en el contexto de que la aplicación de ámbito industrial que va a ser portada almacena su información en un servidor web con el que se puede efectuar peticiones HTTP y así adquirir los datos que sean necesarios.

De este modo, no necesitamos almacenar en el sistema todo el grueso de la información, sino simplemente el mínimo de datos necesarios para realizar peticiones HTTP y procesar la información recibida en formato JSON, obteniendo así los objetos asociados con los que vamos a trabajar. El *GameObject* de la escena que contiene esta funcionalidad es «ServerApiManager»,

Este objeto contiene la clase homónima *ServerApiManager* que gestiona toda esta funcionalidad. Contiene dos variables editables desde el entorno de Unity para asignar un usuario y un password a la hora de hacer peticiones al servidor.

Una vez más se ha aplicado el patrón *singleton* con objeto de facilitar las comunicaciones y para evitar la creación de más de una instancia de esta clase.

Pasando a la lógica de la clase, destacamos los siguientes puntos:

- **Cola de peticiones API:** las peticiones a la API que van llegando desde la aplicación de *HoloLens 2* se van guardando en una cola para ir procesándolas una a una y así evitar solapamientos y otros posibles errores. De este modo, cuando se comienza el procesamiento de una petición se inicia una corrutina y se bloquea el procesamiento de las peticiones de la cola hasta que esta petición no ha terminado de ser procesada.

Listado 5.12: Método RequestsQueueHandler() que utiliza una estructura de tipo cola para gestionar las peticiones al servidor web

```
1   private IEnumerator RequestsQueueHandler()
2   {
3       while (true)
4       {
5           if (processingRequest == false && apiRequestQueue.Count > 0)
6           {
7               processingRequest = true;
8               ApiRequest apiRequest = (ApiRequest)apiRequestQueue.Dequeue();
9               ManageApiRequests(apiRequest);
10          }
11          //Wait until request is done
12          yield return new WaitUntil(() => processingRequest == false);
13      }
14  }
```

Aquí comienza el procesamiento de las peticiones, que únicamente pueden ser de dos tipos:

a. De tipo «JSON», que serían aquellas peticiones de las que se espera como respuesta un archivo JSON.

b. De timo «Image», que son aquellas peticiones que se espera que devuelvan una imagen en formato png.

Para gestionar las peticiones y sus tipos se ha creado una clase llamada *ApiRequest*, la cual contiene la información necesaria para efectuar las peticiones, que es:

- El tipo de petición, para gestionarla correctamente. Para ello se utiliza una clase propia de tipo «ApiRequestType» que almacena los tipos como constantes.
- Los parámetros de la petición, que es un diccionario con los posibles parámetros que han de incluirse en el URL de la petición.
- El URL de la petición.
- El idioma de la cabecera de la petición, para recibir la información en ese idioma.

Listado 5.13: Clase *ApiRequest* que encapsula la lógica relativa a los objetos de peticiones al servidor web

```
1   public class ApiRequest
2   {
3       private string type;
4       private Dictionary<string, string> parameters;
5       private string urlRequest = "";
6       private string languageHeader;
7
8       public ApiRequest(string type, string
            ↪ languageHeader, Dictionary<string, string>
            ↪ parameters = null)
9       {
10          this.type = type;
11          this.parameters = parameters;
12          this.languageHeader = languageHeader;
13
14      }
15  }
```

- **Procesamiento de las peticiones:** la lógica para el procesamiento de las peticiones es la siguiente:

  1. *Comprobación en la caché.* Primero se comprueba en una base de datos de local si ya se tiene almacenado el JSON asociado a esa petición. Esta funcionalidad de caché la gestiona la clase *CacheDatabaseManager*. Si ya se tiene almacenado el JSON, pasaríamos a su procesamiento (que abordaremos en el siguiente bloque) sino, continuamos con el proceso de petición HTTP para obtenerlo.
  2. *Clasificación de la petición HTTP de JSON en función de su tipo.* Para obtener el JSON es necesario realizar una petición HTTP específica. Cabe destacar que pueden existir peticiones de las que no se espera un JSON como respuesta, por ejemplo para una petición de *login*. En el contexto en el que se ha desarrollado esta aplicación hemos contemplado tres posibles tipos de peticiones:
     - **Petición de Login**. Esta petición es de tipo POST y requiere de variables concretas como los datos de usuario (nombre y *password*). Permite el acceso al servidor web para realizar el resto de peticiones.
     - **Petición de Logout**. Petición de tipo POST que no requiere de variables adicionales, únicamente una *cookie* de identificación obtenida en previas peticiones. Efectúa el *logout* del servidor.
     - **Peticiones GET genéricas**. Son el resto de peticiones las cuales van a devolver un JSON con información de los objetos de negocio.

     Para realizar las peticiones en Unity se ha utilizado la biblioteca «UnityWebRequest», que contiene clases y métodos destinados a gestionar este tipo de comunicaciones.
  3. *Procesamiento de la petición HTTP.* Una vez clasificada la petición y enviada al método adecuado para su procesamiento empieza la corrutina encargada de realizar la petición.

     La parte más destacable es la relativa a la construcción de la URL de la petición. En el contexto de desarrollo de esta aplicación, la URL contiene campos entre corchetes "{}" que son sustituidos por las variables de los objetos que se requieren para crear esa URL concreta.

     Toda esta parte de construcción de la URL se gestiona a través de la clase *ServerApiData*, la cual contiene información con las URL de las peticiones y el método *BuildUrlRequest()* para sustituir estos campos entre corchetes por las variables que le pasemos para construir así la URL concreta de la petición.

Listado 5.14: Método *ApiUrlReplacer* que se encarga de formatear las URL con los parámetros adecuados

```
1    public static string ApiUrlReplacer(string url, ↩
         ↪ Dictionary<string, string> parameters)
2    {
3        foreach (KeyValuePair<string, string> ↩
             ↪ parameter in parameters)
4        {
5            url = url.Replace("{" + parameter.Key + ↩
                 ↪ "}", parameter.Value);
6        }
7
8        return url;
9    }
```

Por ejemplo, el esqueleto de una URL almacenado sería:

`https://ordenadores.com/api/web/tipos/{partId}/informacion-tecnica`

Y una vez procesada quedaría así:

`https://ordenadores.com/api/web/tipos/gpu1234ABCD/informacion-tecnica`

Volviendo al procesamiento de las peticiones, una vez tenemos la URL construida, se realiza el envío de la petición y se espera su respuesta. Si no ha habido ningún error se obtiene un archivo JSON que es almacenado en la caché asociándolo a la URL que hemos utilizado. También se llama a una serie de métodos que imprimen y almacenan información relativa a la respuesta HTTP para dejar un registro de las mismas.

Una vez tenemos el JSON, el siguiente paso es procesarlo para obtener los objetos de la lógica de negocio que necesitamos para trabajar con ellos más fácilmente (incluirlos en listas, acceder a sus variables, etc.).

Componentes relativos al procesamiento de los JSON devueltos por el servidor web.

La clase responsable del procesamiento de los Json es *JsonResponsesHandler*. El procesamiento de los JSON empieza con el método de esta clase *HandleJson()*, que recibe como parámetros el tipo de la petición y el JSON en formato string. Este método se encarga de encauzar el flujo del procesamiento en función del subtipo de la petición pues cada uno da lugar a objetos de negocio diferentes. Por lo tanto tenemos un tipo de petición JSON específica para cada objeto de negocio que necesitemos obtener.

Listado 5.15: Método *HandleJson* que se encarga de gestionar los JSON para obtener los objetos de negocio adecuados

```
1     public object HandleJson(string apiRequestType, string json)
2     {
3         switch (apiRequestType)
4         {
5             case ApiRequestType.Marcas:
6                 responseObject = HandleBrandsJson(json);
7                 break;
8
9             case ApiRequestType.Modelos:
10                responseObject = HandleModelsJson(json);
11                break;
12
13            case ApiRequestType.DatosDelSistema:
14                responseObject = HandleSystemDataJson(json);
15                break;
16
17            //
18            //[...]
19            //
20
21            default:
22                break;
23        }
24
25        return responseObject;
26
27    }
```

Para convertir un JSON en un objeto determinado se ha hecho uso de la clase *JsonUtility* y su método *FromJson()*, el cual a partir de un JSON obtiene un objeto de una clase determinada. La clase debe estar presente en los archivos del proyecto con la etiqueta [Serializable] para permitir la creación de objetos a través de este método.

De este modo los JSON son convertidos en objetos o listas de objetos en función de la petición realizada y se devuelven a la clase *ServerApiManager* para continuar con el último paso del proceso.

Finalmente y para acabar este apartado, la clase *ServerApiManager* crea un evento de tipo «OnApiProccesedResponse» con el objeto creado, al cual está suscrita la clase encargada de la comunicación con la aplicación de las *HoloLens 2* (*DesktopDataChannelMessagesManager*). Esta clase recibe el objeto y lo envía por el canal de datos en forma de array de bytes, como en el caso de las otras aplicaciones (Módulo 1 y 2).

Listado 5.16: Método *SendApiProccesedResponse()* que se encarga de enviar el objeto resultado de la petición por el canal de datos hacia la aplicación de HoloLens 2

```
1   private void SendApiProccesedResponse(object apiResponse)
2   {
3       byte[] byteArray = null;
4
5       if (jaltestDataChannel.State == ↩
            ↪ DataChannel.ChannelState.Open)
6       {
7           if (apiResponse is byte[] == false)
8               byteArray = ObjectToByteArray(apiResponse);
9           else
10              byteArray = (byte[])apiResponse;
11
12          jaltestDataChannel.SendMessage(byteArray);
13      }
14  }
```

5.4.2. Aplicación de HoloLens 2

A continuación se expone un *diagrama de clases* simplificado que recoge los subsistemas expuestos previamente de la aplicación de *HoloLens 2* del módulo 3 con sus clases, atributos, métodos y relaciones más significativas.

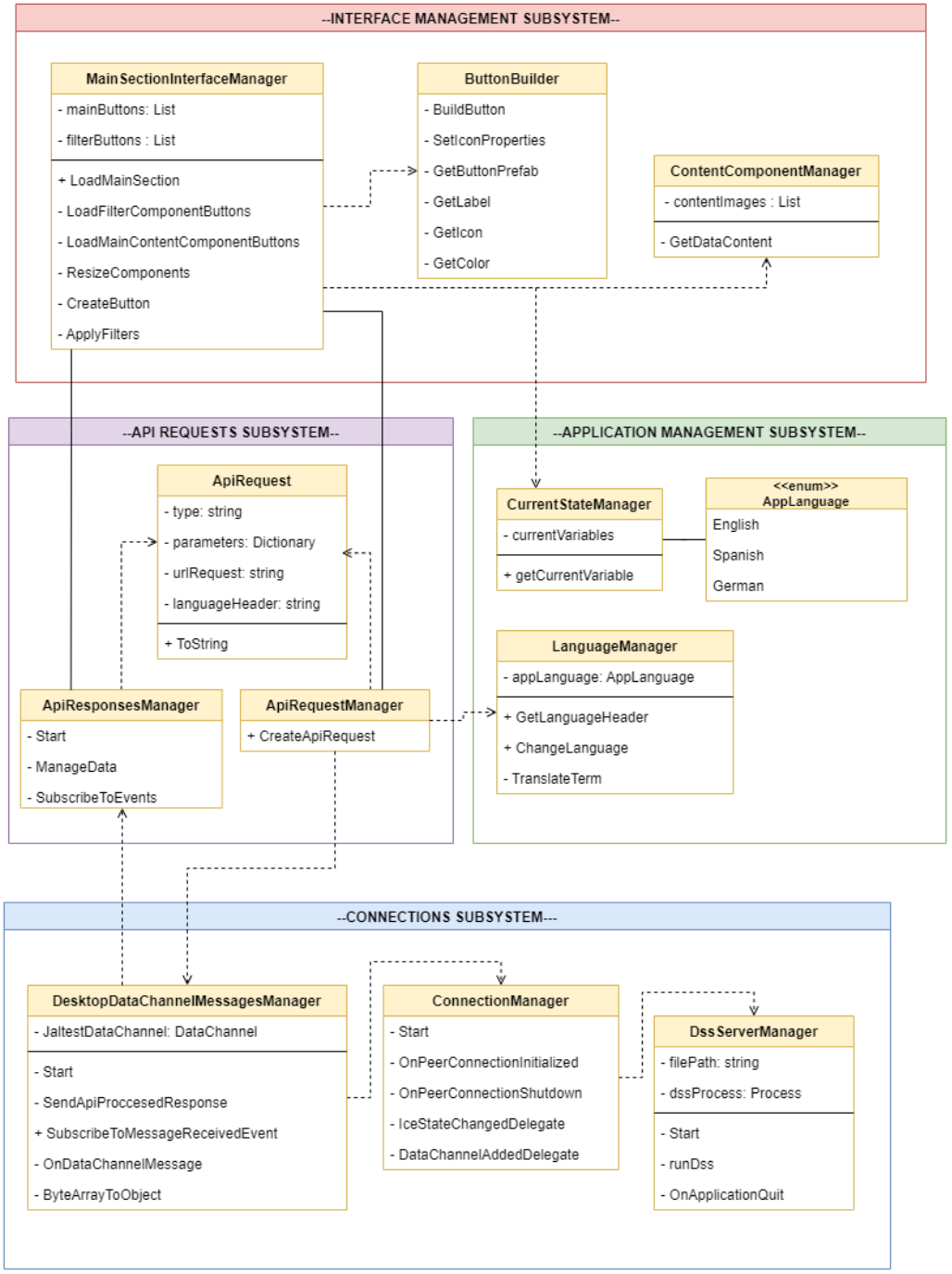

Figura 5.16: Diagrama de clases de la aplicación de *HoloLens 2* del módulo 3)

Para la explicación de la parte del módulo 3 correspondiente a esta aplicación de *HoloLens 2*, se va a hacer una exposición basada en los componentes gráficos y de lógica más representativos.

Componentes gráficos

Los componentes gráficos que se han empleado en este módulo 3 son tanto propios como nativos de MRTK. Los propios han sido construidos para satisfacer las necesidades concretas de la aplicación de ámbito industrial con la que se ha trabajado, pero igualmente pueden ser fácilmente reutilizados para otros proyectos y contextos.

A continuación, se exponen los principales componentes gráficos:

- **Botones genéricos:** los botones son elementos gráficos que inician una acción cuando son pulsados. En esta aplicación se han contemplado diferentes tipos de botón para satisfacer los requisitos que se tenían:
  - Botón básico solo con texto
  - Botón con texto y etiqueta básica
  - Botón con texto e icono
  - Botón con texto, etiqueta e icono
  - Botones para barra de filtro con texto, etiqueta icono e *input* de *check*

  Todos estos botones, excepto los de filtro, se crean a partir del mismo modelo de objeto. En función del tipo de botón, se activarán unos componentes u otros, se elegirán los iconos, los colores, el tamaño y la posición de los elementos, etc. Todo ello se explicará en la parte de la lógica.

  Un botón de filtro es idéntico salvo por la diferencia de que incluye dos iconos para la imagen de *check* (marcado y sin marcar).
- **Botones con imágenes:** son botones que en vez de texto contienen una imagen. La estructura es muy similar a la de los otros botones. Además puede tener texto también. En el caso de la imagen el texto aparece cuando haces fijas la vista en el botón.
- **Ventanas contenedoras:** es un componente nativo de MRTK que facilita la creación rápida de ventanas para contener otros elementos gráficos. Está compuesta por una barra de título con texto y dos botones para cerrarla y otro para activar la función de *autoseguimiento*.

  En definitiva se trata de una ventana básica, pero a la que se puede añadir funcionalidades como la de ocultar los elementos que salgan de sus márgenes.
- **Contenedores de imágenes:** son componentes que permiten cargar imágenes para ser mostradas. Las imágenes se aplican como texturas sobre el mismo material.
- **Bloques de contenido:** es un tipo especial de ventana contenedora diseñada para cargar contenido de texto e imágenes.
- **Menú horizontal:** menú que contiene botones dispuestos de manera horizontal. Permite el desplazamiento por control gestual o con los botones de *scroll*. En la aplicación implementada se utiliza para contener los botones de tipo filtro.
- **Menú vertical:** menú que contiene botones dispuestos de manera vertical. Permite el desplazamiento por control gestual o con los botones de *scroll*. En la aplicación implementada se utiliza para contener los botones estándar.

Componentes relativos a la lógica

En este punto se destacarán las clases más importantes responsables de la lógica de esta aplicación

- **Mánager de peticiones:** la clase responsable de la gestión de las peticiones al servidor web se llama *ApiRequestManager*. Esta clase contiene los métodos específicos para realizar peticiones concretas para los objetos de negocio necesarios para construir la interfaz de la aplicación y mostrar al trabajador la información que precisa. Todos los métodos tienen la misma estructura:
  1. Reciben los parámetros necesarios para construir la petición.
  2. Construyen la petición, con un diccionario de parámetros y una variable para definir el tipo de petición.
  3. Envían la petición a la clase gestora del canal de datos *HoloDataChannelMessageManager* para que la mande por el canal de datos hacia la aplicación de escritorio.
- **Mánager de respuestas:** clase responsable de gestionar las respuestas del servidor que llegan a través del canal de datos. El nombre de la clase es *ApiResponsesManager*. Esta clase tiene dos bloques fundamentales:
  - Uno es el destinado a suscribirse a los eventos de respuesta de peticiones (uno por cada tipo).
  - El otro bloque es el de los métodos para gestionar cada respuesta. Los métodos de gestión de las respuestas se encargan de obtener los objetos necesarios para la construcción de la interfaz, utilizando clases complementarias para ello (para construir los botones, obtener otra información etc.). Cuando el método en cuestión tiene todos los objetos, llama al gestor del componente principal de la aplicación (el que muestra los menús y contenido principal) para que construya la interfaz con los objetos necesarios.

Un ejemplo de método gestor de respuesta sería el siguiente:

Listado 5.17: Método *ManageBrandsData()* que se encarga de gestionar la lista de objetos de negocio «brands» obtenida como resultado de una petición al servidor web

```
1   private void ManageBrandsData(Brand[] brands)
2   {
3       ButtonProperties buttonProperties;
4       List<ButtonProperties> buttonPropertiesList = new ↩
            ↪ List<ButtonProperties>();
5
6       foreach (Brand brand in brands)
7       {
8           buttonProperties = new ↩
                ↪ ButtonProperties(ButtonType.
9           MainComponentButtonSimple, brand.name, brand);
10          buttonPropertiesList.Add(buttonProperties);
11      }
12
13      if (UiOptions.Instance.SortMenuButtonsAlphabetically)
14      {
15          buttonPropertiesList.Sort((x, y) => ↩
                ↪ string.Compare(x.MainText, y.MainText));
16      }
17
18      MainSectionProperties mainSectionProperties = new ↩
            ↪ MainSectionProperties(MainSectionType.Brands, ↩
            ↪ "Brands", buttonPropertiesList);
19      CurrentStateManager.Instance.
20      CurrentBrandSectionProperties = mainSectionProperties;
21
22      MainThreadManager.manager.AddJob(() =>
23      {
24          CurrentStateManager.Instance.LoadMainInterface();
25          MainSectionInterfaceManager.Instance.
26          LoadMainSection(mainSectionProperties);
27      });
28  }
```

- **Constructor de botones:** clase que encapsula la lógica necesaria para construir los botones utilizados en la aplicación. Hace uso de la clase complementaria *ButtonProperties* que define las propiedades de un botón. Sus atributos son

Listado 5.18: Atributos de la clase *ButtonProperties* que encapsula lógica relativa a los botones

```
1   public class ButtonProperties
2   {
3       private string buttonType;
4       private string mainText;
5       private object attachedObject;
6       private string labelText;
7       private string buttonColor;
8       private string icon;
9       private object secondAttachedObject;
10      private List<ButtonProperties> buttonChilds;
11  }
```

En función del tipo de botón «buttonType» la clase *ButtonBuilder* llama al método encargado de construir ese botón con las propiedades que se le pasan. Además *ButtonBuilder* contiene métodos complementarios para obtener los colores e iconos asociados a un botón en concreto que son definidos en clases que contienen estas relaciones.

Por ejemplo, podemos tener una serie de botones para filtros que representen categorías. La categoría "A" puede ser de color rojo y con el icono de un cuadrado. Esta información se guarda en clases concretas para los nombres de las categorías, los colores, los iconos y la asociación de estos tres elementos. De modo que conociendo el nombre del botón (Su categoría en este caso), *ButtonBuilder* es capaz de construir el botón con el color y el icono asociado a esa categoría.

Cabe mencionar también los atributos para asociar objetos al botón. Estos objetos son los que se pasarán al gestor de clicks cuando se pulse el botón, para que en función del tipo de objeto asociado llame a un método u otro.

- **Constructor de la sección principal:** La clase responsable de la construcción de la sección principal es la clase *MainSectionInterfaceManager*. Esta clase a su vez hace uso de la clase *MainSectionProperties* para ayudarse en esta tarea, como en el caso de los botones. Para hacernos una idea de lo que se necesita para construir la sección principal, a continuación se muestra los atributos de esta clase de propiedades:

Listado 5.19: Atributos de la clase *MainSectionProperties*

```
1  public class MainSectionProperties
2  {
3      private string mainSectionType;
4      private string mainSectionTittleBarText;
5      private List<ButtonProperties> ↩
           ↪ mainContentButtonsPropertiesList;
6      private List<ButtonProperties> ↩
           ↪ filterButtonsPropertiesList;
7  }
```

El atributo «mainSectionType» define el tipo de la sección principal. Estos tipos se han definido para tener varias configuraciones posibles con diferentes tamaños de los botones, diferentes configuraciones de elementos visibles de la interfaz, etc.

El «atributomainSectionTittleBarText» sirve para definir el título de la barra de la sección principal.

Finalmente tenemos las dos listas de propiedades de botón, una para el componente de filtro (menú superior) y otra para el componente de botones principales (menú de la sección principal), porque así fue la configuración diseñada para esta aplicación. Cada elemento de propiedades de esta lista sirve para construir los botones de cada componente a través de la clase *ButtonBuilder* explicada anteriormente.

- **Constructor del componente de contenido**: La clase responsable de construir el componente de contenido es *ContentComponentManager*. Esta clase encapsula toda la funcionalidad necesaria para crear los hologramas que almacenan la información que necesitan los técnicos para hacer sus tareas de reparación o mantenimiento. Tiene tres funciones principales:

  1. Cargar y situar correctamente las líneas de texto.
  2. Cargas y situar correctamente las imágenes.
  3. Añadir estos elementos al objeto «ClippingBounds» para ocultar los elementos que se salgan de los márgenes de la ventana.

  Además, se ha implementando funcionalidad de *zoom* y *scroll* para esta ventana permitiendo tratar el contenido de la misma como si de un documento PDF se tratara.

CAPÍTULO 6

Conclusiones

6.1. COSTES

6.1.1. Costes temporales

El desarrollo del proyecto enmarcado en este TFG dio comienzo el día 13 de diciembre de 2021 y finalizó el 1 de Julio de 2022. La dedicación media ha sido de 25 horas semanales. Las cifras desglosadas son:

- 28.8 semanas (28 semanas y 4 días).
- 144 días.
- 720 horas.

Para la elaboración de la memoria se estima una dedicación de 150 horas.

6.1.2. Costes económicos

Asumiendo un sueldo de 37,32€ [1] la hora, sin incluir el tiempo dedicado a la redacción de la memoria, los costes son los siguientes:

Tabla 6.1: Tabla con los costes desglosados del proyecto

Descripción	Cantidad	Coste
Sueldo desarrollador	1	26870,4 €
Microsoft HoloLens 2	1	3631€
Ordenador para el desarrollo	1	1336,65 €
Total		**31838,05€**

[1] https://www.payscale.com/research/US/Job=Unity_Developer/Salary/

6.2. ESTADÍSTICAS DEL REPOSITORIO

Se han realizado un total de 152 commits, de los cuales 131 son de mi autoría y 21 fueron realizados por mi compañero en el proyecto.

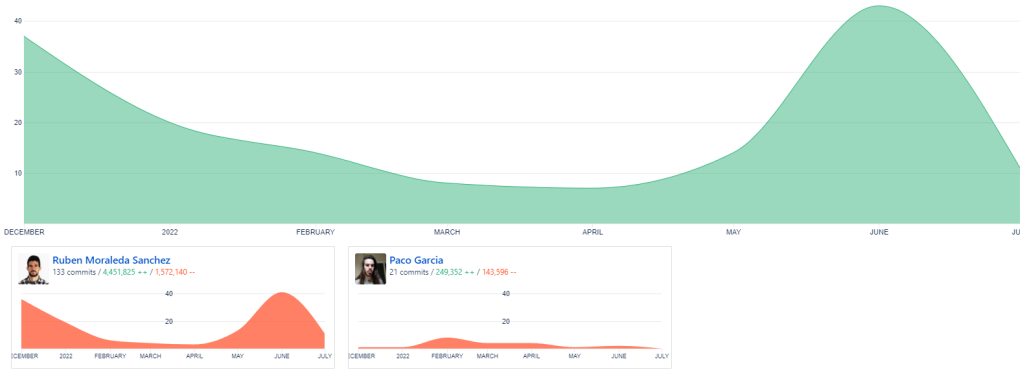

Figura 6.1: Estadísticas del repositorio

Utilizando la herramienta *cloc*[2] sobre los directorios del código fuente del proyecto se han obtenido las siguientes estadísticas:

Tabla 6.2: Estadísticas sobre el código fuente del proyecto

Lenguaje	Archivos	Comentarios	Líneas de código
C#	139	676	8016

En la siguiente imagen se puede ver la distribución de commits en función del día de la semana y la hora, pudiéndose comprobar que se ha concentrado el trabajo durante las horas laborales y la franja tanto de mañana como de tarde (obtenido de *Bitbucket*[3])

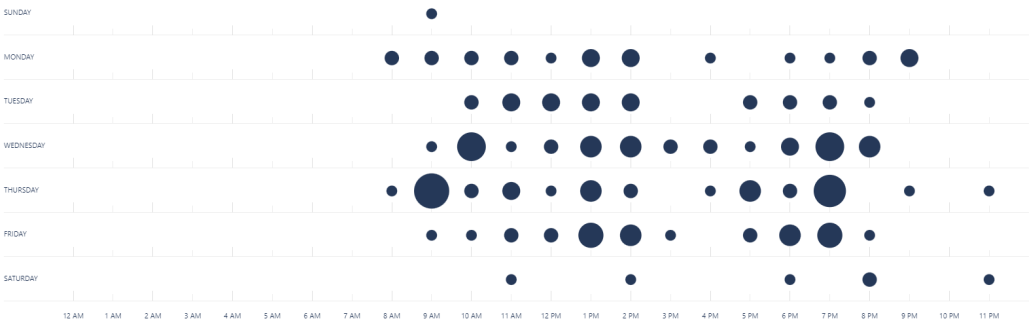

Figura 6.2: Distribución de commits por días y horas

[2] https://github.com/AlDanial/cloc/
[3] https://bitbucket.org/

6.3. JUSTIFICACIÓN DE COMPETENCIAS ADQUIRIDAS

En esta sección se incluye una tabla con las competencias de la intensificación de Ingeniería del Software que han sido adquiridas, practicadas y reforzadas durante la elaboración del presente Trabajo Fin de Grado.

Tabla 6.3: Tabla de competencias adquiridas relativas a la intensificación de Ingeniería del Software

Competencia	Justificación
[IS1] Capacidad para desarrollar, mantener y evaluar servicios y sistemas software que satisfagan todos los requisitos del usuario y se comporten de forma fiable y eficiente, sean asequibles de desarrollar y mantener y cumplan normas de calidad, aplicando las teorías, principios, métodos y prácticas de la Ingeniería del Software	El desarrollo del sistema software desarrollado en este TFG ha atendido a criterios de calidad, eficiencia, fiabilidad y se ha vigilado que las soluciones elegidas sean óptimas en cuanto a su mantenimiento y mejora.
[IS3] Capacidad de dar solución a problemas de integración en función de las estrategias, estándares y tecnologías disponibles.	Uno de los aspectos más importantes en este TFG es el de la integración de diferentes sistemas, para lo cual se ha debido de realizar un trabajo de investigación y elección de las tecnologías disponibles más adecuadas para este propósito.
[IS4] Capacidad de identificar y analizar problemas y diseñar, desarrollar, implementar, verificar y documentar soluciones software sobre la base de un conocimiento adecuado de las teorías, modelos y técnicas actuales	Durante el desarrollo del TFG han surgido problemas, retos y situaciones nuevas que se han afrontado utilizando el conocimiento adquirido en la carrera y especialmente en los últimos años de la intensificación, utilizando patrones, modelos y técnicas propias de la Ingeniería del Software.

6.4. CONCLUSIÓN DEL PROYECTO

A continuación se revisarán los objetivos del proyecto expuestos en el capítulo 3 para indicar su grado de completitud al término del mismo.

6.4.1. Objetivo general

El objetivo general era la construcción de una plataforma compuesta por tres aplicaciones para facilitar la convergencia y la conversión de aplicaciones estándar de ordenador al entorno de la realidad mixta.

Este objetivo se ha visto alcanzado pues las tres aplicaciones que se propusieron se han desarrollado: La aplicación de captura de pantalla y streaming, la aplicación de soporte por videoconferencia y la aplicación de soporte a partir de componentes nativos de MRTK.

6.4.2. Usabilidad: el sistema deberá ser usable.

En general, se puede afirmar que se ha alcanzado un alto grado de usabilidad de las diferentes aplicaciones. A pesar de tratarse de una tecnología muy reciente y que posee margen de mejora en este sentido, el manejo de los hologramas y la interacción con los diferentes componentes gráficos se realiza sin entrañar mucha dificultad, y una persona sin experiencia es capaz de familiarizarse rápidamente con el uso de estas aplicaciones.

6.4.3. Empleo de diferentes canales de interacción: la interacción con el sistema deberá basarse en gestos y comandos de voz

No solo se ha utilizado la interacción gestual que brinda el MRTK por defecto, sino que además se han diseñado e implementado tipos de gestos nuevos para este propósito (para los diferentes clicks, arrastrar, etc.), ampliando la variedad de los gestos.

Respecto a los comandos de voz, también han sido trabajados e implementados, encontrando soluciones a problemas en la dicción de términos en inglés y configurando todos los comandos que han sido necesarios y útiles.

6.4.4. Mantenibilidad y escalabilidad: el sistema deberá construirse atendiendo a criterios de mantenibilidad y escalabilidad

La naturaleza de los tres módulos/aplicaciones construidas permiten una fácil mantenibilidad y escalabilidad, por las siguientes razones:

- El módulo de Streaming puede utilizarse con cualquier aplicación de escritorio debido a su sencillez, y si se necesita ampliar funciones o tipos de interacción también es muy fácil añadirlas gracias al patrón Object de Unity. Las mismas razones se aplicarían al módulo de videollamada.
- Por último, la aplicación construida con componentes nativos, al estar desarrollada a través de elementos gráficos fácilmente reutilizables, y estar su lógica bien separada y delimitada atendiendo a criterios de responsabilidad entre los diferentes objetos de la escena, la mantenibilidad y la escalabilidad se ven ampliamente garantizadas.

6.4.5. Sistema plurilingüe: el sistema deberá soportar varios idiomas

En el módulo 3 se implementó un sistema para traducir los textos de la aplicación a diferentes idiomas, en concreto Español, Inglés y Alemán (que son los soportados por la aplicación de ámbito industrial con la que se ha trabajado).

Por lo tanto, podemos afirmar que este objetivo se ha visto plenamente alcanzando.

6.4.6. Delegación de tareas y comunicación: el sistema deberá desarrollarse siguiendo una arquitectura cliente-servidor

La arquitectura cliente-servidor ha sido implementada en el módulo 3 para la aplicación nativa, ya que en el resto no tiene sentido. Las operaciones más demandantes y todos los cálculos que se han podido delegar se han delegado en el ordenador de escritorio, agilizando así el rendimiento de las HoloLens 2.

6.4.7. Estabilidad del sistema y recuperación de errores: el sistema deberá ser capaz de recuperarse ante errores

Los errores más frecuentes y graves son controlados adecuadamente con las estructuras habituales try-catch, impidiendo así la brusca detención del sistema en la mayoría de casos.

6.4.8. Uso de tecnologías y estándares libres: el sistema deberá ser desarrollado mediante el uso de herramientas libres

Todos los módulos han sido desarrollados con estándares y tecnologías libres. No se ha necesitado comprar ninguna licencia, pues todas son libres, ni tampoco se ha tenido que invertir en herramientas de pago.

6.4.9. Alta optimización: el sistema deberá estar fuertemente optimizado

En el desarrollo del sistema se ha prestado especial atención a estos aspectos, utilizando patrones que aseguran soluciones basadas en las mejores formas conocidas para problemas determinados, como el patrón *listener* o el gestor de colas.

También el uso de la caché para el módulo 3 es otro ejemplo de mejora en la optimización del sistema.

6.5. TRABAJO FUTURO

Durante el proceso de diseño y construcción de la plataforma *Apholo* fueron surgiendo ideas sobre mejoras y nueva funcionalidad que no ha sido posible implementar en este periodo, pero que sería interesante como un posible trabajo en el futuro. Algunas de estas mejoras son las siguientes:

A) **Autodetección de la pareja de dispositivos en la misma red local para iniciar la llamada de forma automática al arrancar las aplicaciones**. Esta función evitaría el tener un botón para realizar la llamada en la aplicación de *streaming* y en la nativa, y en el caso de la aplicación nativa ahorraría también toda la parte de la interfaz de la aplicación de escritorio, de modo que esta ventana podría mantenerse oculta.

B) **Mejorar la funcionalidad de *scroll* en la aplicación de *streaming***. Aunque la solución de los botones permite realizar correctamente la función de *scroll*, lo ideal sería contar con un gesto para dicha acción de modo que toda la interacción del mismo tipo se realice a través de gestos y no utilizando canales diferentes.

C) **Añadir sonidos y elementos visuales que den mayor *feedback* sobre la interacción**. Sería una mejora muy buena para la usabilidad el añadir sonidos o efectos visuales a las diferentes interacciones para reportar información al usuario sobre el éxito de sus acciones sobre el sistema.

D) **Agregar la funcionalidad del módulo 2 de videollamada al módulo de *streaming* y al módulo nativo**. Aunque la función de videollamada tiene utilidad por si sola, también resultaría muy útil y se sentiría muy natural su implementación como una función adicional dentro de las aplicaciones de *streaming* y la elaborada con componentes nativos de MRTK, ya que su función es la de ofrecer soporte y la videollamada constituiría un canal adicional para este propósito.

6.6. CONCLUSIÓN PERSONAL

El desarrollo de este Trabajo Fin de Grado ha supuesto un antes y un después en mi formación como Ingeniero Informático, por muchos motivos.

En primer lugar, porque he podido aplicar los conocimientos adquiridos en la carrera en un entorno laboral real. Este entorno te obliga en cierta manera a ser mucho más eficiente, responsable y profesional en tu labor, porque ya no solo respondes ante ti mismo como en el ámbito académico, sino que de ti dependen otros compañeros y el futuro del proyecto en el que estás involucrado.

Relacionado con el punto anterior, también el trabajar con una metodología específica, atendiendo a fechas, requisitos, reuniones, tablones *Kanban*, etc., es sin duda un salto cualitativo y constituye una gran experiencia enriquecedora el poder aplicar todos estos conceptos aprendidos en la carrera.

Por otra parte, también he de destacar que haber realizado este TFG en la spin-off Furious Koalas me ha ayudado en cierta forma a que el salto del mundo académico al laboral no sea tan brusco, ya que conocía a algunos de los profesores que en este periodo se han convertido en compañeros de trabajo, y también, se percibe en esta empresa ese ambiente dinámico y fresco de la universidad con el que ya estoy familiarizado.

Por último, también destaco que, el enfrentarme en muchas ocasiones a la incertidumbre, a nuevos problemas y retos para los que no tenía una solución clara en un principio, sin duda me ha servido para mejorar mis habilidades como ingeniero informático, buscando soluciones creativas y aumentando mi sentimiento de autoeficacia en la realización de las tareas que como futuro profesional de la informática debo realizar.

Ha sido este, por lo tanto, un periodo muy exigente, pero a la vez muy enriquecedor, y del que quedo plenamente satisfecho y contento.

Bibliografía

[1] Klaus Ahlers, David E. Breen, Chris Crampton, Eric Rose, Mihran Tuceryan, Ross T. Whitaker, and Douglas S. Greer. An augmented vision system for industrial applications. *SPIE Proceedings*, 1995.

[2] Reinhold Behringer, Steven Chen, Venkataraman Sundareswaran, Kenneth Wang, and Marius Vassiliou. A distributed device diagnostics system utilizing augmented reality and 3d audio. *Eurographics*, page 105–114, 1999.

[3] Eleonora Bottani and Giuseppe Vignali. Augmented reality technology in the manufacturing industry: A review of the last decade. *IISE Transactions*, 51(3):284–310, 2019.

[4] P. Boulanger. Application of augmented reality to industrial tele-training. *First Canadian Conference on Computer and Robot Vision, 2004. Proceedings.*, 2004.

[5] Tara J. Brigham. Reality check: Basics of augmented, virtual, and mixed reality. *Medical Reference Services Quarterly*, 36(2):171–178, 2017.

[6] Britannica. Education and training. URL: https://www.britannica.com/technology/virtual-reality/Education-and-training.

[7] Jack C. Cheng, Keyu Chen, and Weiwei Chen. State-of-the-art review on mixed reality applications in the aeco industry. *Journal of Construction Engineering and Management*, 146(2), 2020.

[8] Kyung H. Chung, John P. Shewchuk, and Robert C. Williges. An application of augmented reality to thickness inspection. *Human Factors and Ergonomics in Manufacturing*, 9(4):331–342, 1999.

[9] Sanika Doolani, Callen Wessels, Varun Kanal, Christos Sevastopoulos, Ashish Jaiswal, Harish Nambiappan, and Fillia Makedon. A review of extended reality (xr) technologies for manufacturing training. *Technologies*, 8(4):77, 2020.

[10] Steven Feiner, Blair Macintyre, and Dorée Seligmann. Knowledge-based augmented reality. *Communications of the ACM*, 36(7):53–62, 1993.

[11] Lipson H, Shpitalni M, Kimura M, and Goncharenko I. *Online product maintenance by web–based augmented reality*. Conference on New Tools and Workflows for Product Development, 1998.

[12] Larry F. Hodges, Barbara Olasov Rothbaum, Renato Alarcoh, David Ready, Fran Shahar, Ken Graap, Jarrell Pair, Philip Hebert, Dave Gotz, Brian Wills, and et al. Virtual vietnam: A virtual environment for the treatment of vietnam war veterans with posttraumatic stress disorder. *PsycEXTRA Dataset*, 1999.

[13] G. Klinker, O. Creighton, A.H. Dutoit, R. Kobylinski, C. Vilsmeier, and B. Brugge. Augmented maintenance of powerplants: A prototyping case study of a mobile ar system. *Proceedings IEEE and ACM International Symposium on Augmented Reality*, 2001.

[14] Myron W. Krueger, Thomas Gionfriddo, and Katrin Hinrichsen. Videoplace—an artificial reality. *Proceedings of the SIGCHI conference on Human factors in computing systems - CHI '85*, 1985.

[15] Shaun W. Lawson and John R. Pretlove. Augmented reality for underground pipe inspection and maintenance. *Telemanipulator and Telepresence Technologies V*, 1998.

[16] Harold A. Layer. Holographic and stereoscopic space: New research directions. *Leonardo*, 22(3/4):411, 1989.

[17] Sergo Martirosov and Pavel Kopecek. Virtual reality and its influence on training and education - literature review. *DAAAM Proceedings*, page 0708–0717, 2017.

[18] Thomas Watt Maver. *Envisioning architecture: Space, time, meaning*. Mackintosh School of Architecture and the School of Simulation and Visaulization at the Glasgow School of Art, 2017.

[19] W. Moczulski, W. Panfil, M. Januszka, and G. Mikulski. Applications of augmented reality in machinery design, maintenance and diagnostics. *Recent Advances in Mechatronics*, page 52–56, 2006.

[20] L.J. Najjar, J.C. Thompson, and J.J. Ockerman. A wearable computer for quality assurance inspectors in a food processing plant. *Digest of Papers. First International Symposium on Wearable Computers*, 1997.

[21] T. Nakagawa, T. Sano, and Y. Nakatani. Plant maintenance support system by augmented reality. *IEEE SMC'99 Conference Proceedings. 1999 IEEE International Conference on Systems, Man, and Cybernetics (Cat. No.99CH37028)*, 1999.

[22] C. Nakajima and N. Itho. A support system for maintenance training by augmented reality. *12th International Conference on Image Analysis and Processing, 2003.Proceedings.*, 2003.

[23] U. Neumann and A. Majoros. Cognitive, performance, and systems issues for augmented reality applications in manufacturing and maintenance. *Proceedings. IEEE 1998 Virtual Reality Annual International Symposium (Cat. No.98CB36180)*, 1998.

[24] Robert Nystrom. *Game Programming Patterns*. Packt, 2014.

[25] History of information. Pygmalion's spectacles. URL: https://www.historyofinformation.com/detail.php?entryid=4543.

[26] AVT simulation. The history of ar vr and mr and build of technology: Avt simulation. URL: https://www.avtsim.com/THE-HISTORY-OF-AR-VR, Oct 2021.

[27] C. Stapleton, C. Hughes, M. Moshell, P. Micikevicius, and M. Altman. Applying mixed reality to entertainment. *Computer*, 35(12):122–124, 2002.

[28] Jonathan Strickland. How virtual reality works. URL: https://electronics.howstuffworks.com/gadgets/other-gadgets/virtual-reality.htm#pt8, Jun 2007.

[29] Day Translations. The story of virtual reality. URL: https://www.daytranslations.com/blog/story-virtual-reality, Jul 2019.

[30] E.C. Urban. *The information warrior*. IEEE spectrum, 1995.

[31] VRS.org. Virtuality – a new reality of promise, two decades too soon. URL: https://www.vrs.org.uk/dr-jonathan-walden-virtuality-new-reality-promise-two-decades-soon, Apr 2018.

[32] Nicholas J Wade. Charles wheatstone (1802 – 1875). *Perception*, 31(3):265–272, 2002.

[33] Wikipedia. Stereopsis. URL: https://en.wikipedia.org/wiki/Stereopsis.

[34] M. L. Yuan, S. K. Ong, and Andrew Y. Nee. The virtual interaction panel: An easy control tool in augmented reality systems. *Computer Animation and Virtual Worlds*, 15(34):425–432, 2004.