TecnoCampus
Escola Superior Politècnica

Centre adscrit a la
upf. Universitat Pompeu Fabra Barcelona

**Doble Titulación en Grado en Informática de Gestión y Sistemas de Información y Grado en Diseño y Producción de Videojuegos**

# Gemelos Digitales y Mundos Virtuales: Impulsando la Transformación del Puesto de Trabajo

**Memoria**

**REBECA GARCÍA FRANCO**
**TUTORES: DR. LÉONARD JANER**
**DR. ENRIC SESA I NOGUERAS**

2022/23

TecnoCampus
Mataró-Maresme

# Dedicatoria

A mi familia, amigos y a ti, por la paciencia y el apoyo durante todo este proceso.

A la persona que me saco de las malas, que ha confiado siempre en mí más que yo. Sin ti no estaría aquí.

A mi tutor Léonard, allá donde este, sin él no hubiese sido posible.

# Agradecimientos

A mis tutores, Enric Sesa y Léonard Janer, por hacer que salga adelante este proyecto.

A mis compañeros de innovación en Avanade, por ser el mejor equipo de trabajo.

A mi mentora de 3D, sin ti seguiría modelando a día de hoy.

# Abstract

This work explores the intersection of digital twins and real-time connected virtual worlds as technology for optimizing working environments. With the advent of digital transformation and advances in disciplines such as the Internet of Things, new working structures and partnership models have emerged that offer interesting proposals. In collaboration with the Innovation team of Avanade, a technology consultancy, this project develops a tailored virtual world using a game engine, which allows real-time communication with a digital twin to simulate, interact and optimize the physical processes and productivity of its Barcelona office.

# Resum

En aquest treball s'explora la intersecció dels bessons digitals i els mons virtuals connectats en temps real com a tecnologia per a optimitzar els entorns de treball. Amb l'arribada de la transformació digital i els avanços de disciplines com *Internet of Things*, han sorgit noves estructures de treball i models de col·laboració que brinden interessants propostes. En col·laboració amb l'equip de innovació de l'empresa Avanade, una consultoria tecnològica, aquest projecte desenvolupa un món virtual a mida utilitzant un motor de jocs, que permet la comunicació en temps real amb un bessó digital per a simular, interactuar  i optimitzar els processos físics i la productivitat de la seva oficina de Barcelona.

# Resumen

En este trabajo se explora la intersección de los gemelos digitales y los mundos virtuales conectados en tiempo real como tecnología para optimizar los entornos de trabajo. Con la trans. Con la llegada de la transformación digital y los avances de disciplinas como el *Internet of Things*, han surgido nuevas estructuras de trabajo y modelos de colaboración que brindan interesantes propuestas. En colaboración con el equipo de innovación de la empresa Avanade, una consultoría tecnológica, este proyecto desarrolla un mundo virtual a medida utilizando un motor de juegos, que permite la comunicación en tiempo real con un gemelo digital para simular y optimizar los procesos físicos y la productividad de su oficina de Barcelona.

# Índice

# Índice de figuras

# Índice de tablas

# Glosario de términos y acrónimos

| | |
|---|---|
| AAD | *Azure Active Directory*. Servicio de gestión de accesos de Azure que permite administrar el acceso a los recursos habilitando la autenticación. |
| API | *Application Programming Interface*. Conjunto de funciones expuestas a través de la definición de *endpoints* para que otras aplicaciones externas interactúen con el sistema. |
| Azure | Plataforma proveedora de servicios *cloud* de Microsoft. |
| *Backlog* | Lista priorizada de funciones y problemas que deben abordarse en futuras iteraciones del desarrollo de un producto. |
| DDD | *Domain-Driven Design.* Enfoque de Desarrollo de software que centra la creación de aplicaciones alrededor del dominio, donde está definida la lógica de negocio. |
| DLL | *Dynamic-Link Library*. Biblioteca compartida de recursos y código que puede ser cargada y ejecutada por varios programas a la vez. |
| DT | *Digital Twin*. Réplica digital de un activo físico que puede utilizarse para la supervisión, el análisis, la simulación y la optimización. |
| Git | Software de control de versiones distribuido que permite el seguimiento de cambios en el desarrollo de un proyecto. |
| GLB | Formato de archivo binario utilizado para almacenar modelos y escenas 3D que combina tanto las geometrías como las texturas asociadas en un único archivo. |
| IA | *Artificial Intelligence*. Simulación de procesos de inteligencia humana por parte de una máquina. |

| | |
|---|---|
| IoT | *Internet of Things*. Movimiento que consiste en la implementación de sensores en los objetos cotidianos para que puedan conectarse e intercambiar datos. |
| JSON | *JavaScript Object Notation*. Formato de intercambio de datos ampliamente utilizado que se representa en formato legible y fácil de entender a través de pares clave-valor organizados en estructura jerárquica. |
| PBI | *Product Backlog Item*. Una unidad de trabajo identificada en el *backlog* de un producto que debe completarse como parte del proceso ágil de desarrollo. |
| *Prefab* | En Unity, es un objeto reutilizable que puede ser instanciado múltiples veces en la misma escena. |
| TDD | *Test-Driven Design*. Proceso de desarrollo de software en el que las pruebas se escriben antes que el código, con el objetivo de garantizar que es correcto según los requisitos. |
| UV | Coordenadas de textura. Coordenadas bidimensionales que se utilizan para mapear y envolver texturas en modelos digitales tridimensionales. |
| VR | *Virtual Reality*. Simulación generada por ordenador de un entorno tridimensional con el que interactuar de forma aparentemente real. |

# 1. Introducción

"Los gemelos digitales en tiempo real cambiarán para siempre la forma en la que vivimos, trabajamos, compramos e impactamos positivamente en nuestro planeta, representando un componente significativo de lo que a menudo se conoce como el metaverso." – John Riccitiello, CEO de Unity [1].

## 1.1. Motivación

Es una realidad que el mundo se encuentra en la era de la transformación digital. Las empresas están pasando a modelos digitales de operaciones, productos y servicios. Y esta transición está impulsada por avances de la tecnología digital, como el Internet de las Cosas, la inteligencia artificial y la computación en el *cloud*. Esto revoluciona la forma en la que la gente aprende, trabaja y se comunica. Esta transformación tiene el potencial de crear una mayor eficiencia, una mejor experiencia para clientes y trabajadores y nuevas fuentes de ingresos todavía inconcebibles.

Esto, potenciado por la pandemia y el trabajo en remoto, brindan una oportunidad única para que nuevas técnicas se puedan incorporar en el espacio de trabajo y hacer de este un nuevo concepto. Una de ellas: el metaverso industrial.

El metaverso representa una nueva frontera en la forma en que interactuamos con la tecnología y entre nosotros. Básicamente consiste en un mundo virtual que existe enteramente en el ámbito digital y permite nuevas formas de comunicación, colaboración e interacción social. Abre nuevas maneras de crear y compartir contenidos, posibilitando formas innovadoras de creatividad y autoexpresión. Las posibilidades son infinitas.

Esto, aplicado al espacio de trabajo, permite reconsiderar los modelos tradicionales, aprovechando los entornos virtuales para ofrecer a las empresas y a sus trabajadores nuevas estructuras de trabajo.

Pero esta revolución no sería posible sin la existencia de otras tecnologías como los digital *twins* y los dispositivos *IoT (Internet of Things)*, entre muchas otras. Esto abre la posibilidad de la supervisión remota, el mantenimiento predictivo, la planificación anticipada y la toma de decisiones en tiempo real.

## 1.2. Objeto y contexto del proyecto

Ante el vigente incremento de las tecnologías incorporadas, este proyecto tiene como propósito, haciendo uso del metaverso, crear un entorno accesible para todos los usuarios y, haciendo uso del gemelo digital, crear una representación digital de objetos y procesos físicos de una oficina.

Se integran las tecnologías de Internet de las Cosas, Gemelos Digitales, Análisis en Tiempo Real, Realidad Virtual y Metaverso Industrial para crear una solución de ingeniería innovadora que facilite las interacciones en un entorno virtual con el objetivo de solucionar problemáticas de la vida real. Así, se podrá probar este nuevo concepto para analizar el bienestar y la optimización de una oficina existente.

El proyecto se va a hacer en colaboración con Avanade, una consultoría tecnológica global que ofrece soluciones digitales. En específico, se trabajará con el equipo de Innovación dentro de Avanade Iberia. Ellos ya contaban con una primera versión de gemelo digital de su oficina creada en Azure, en el que la alumna participó el año pasado. A continuación, se adjunta el grafo de su modelo:

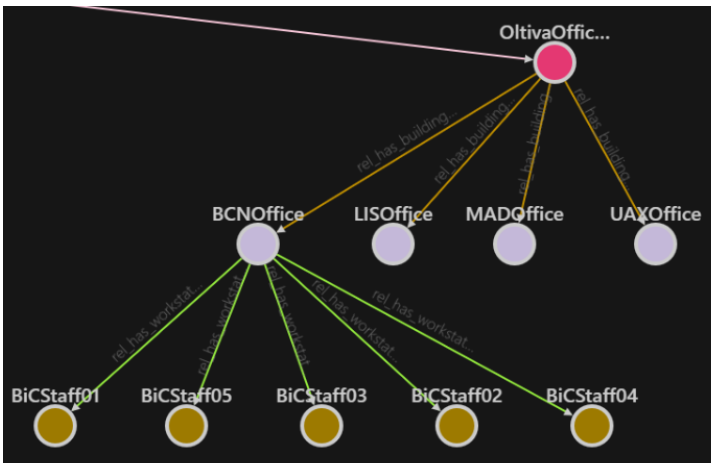


Figura 1.1. Gemelo digital de la oficina de Avanade.

# 2. Marco teórico y análisis de los referentes

## 2.1. Estado del arte

La primera vez que se habla del concepto de *digital twin*, también conocidos como DT, es en 2002, en una presentación de Michael Grieves en la universidad de Michigan. Él lo describió como "una representación virtual de un objeto o sistema físico a lo largo de su ciclo de vida mediante simulaciones" [2]. En aquel momento era algo inmaduro al cual le faltaba base tecnológica, pero la definición ya contenía los componentes que se conocen hoy en día como *digital twin*, incluidos el espacio físico, el espacio virtual y los mecanismos de transferencia de información entre ambos.

A pesar de ser un concepto antiguo, no se empezó a adoptar hasta hace poco, a pesar de que sí que hubo algunas tentativas. La primera empresa que utilizó este concepto en el ámbito laboral fue General Electric, en 2015, año en el cual se lanzó GE Predix Platform [3], la cual permitía utilizar DT para ayudar a sus clientes a supervisar y analizar sus activos industriales. Ellos introdujeron un aspecto muy importante a los DT, y es la recolección de datos a través de medidores y sensores, haciendo uso de la tecnología IoT.

A partir de ahí, muchas de las grandes tecnológicas han ido sumándose. La primera en adoptarlo y defenderlo fue IBM, en 2015, con su plataforma IBM Watson IoT Platform [4]. Más adelante, en junio de 2020, lanzaron su iniciativa IBM Digital Twin Exchange [5]. Después le siguió Microsoft con su plataforma Azure Digital Twins, también en 2020 [6].

Por lo que a metaverso se refiere, la primera vez que se populariza el término es en 1992, gracias a la novela de ciencia ficción llamada *Snow Crash*, escrita por Neal Stephenson. En el libro, el metaverso representaba un mundo virtual que existía en paralelo al mundo real y al cual se accedía a través de internet [7]. Esta definición es muy parecida al concepto actual de este.

El desarrollo de tecnologías avanzadas como la realidad virtual ha hecho más factible la materialización del concepto. Si bien es cierto, el márquetin de algunas grandes empresas, en especial Meta [8], por hacer del metaverso un lugar social más que funcional, ha provocado que la mayoría de los consumidores malinterpreten el término generalizándolo solo a este caso de uso.

El metaverso industrial esconde posibilidades en todos los sectores. Ya se ha utilizado, por ejemplo, para que los clientes potenciales experimenten las mercancías antes de comprarlas, como Ikea con su VR Experience [9], pero esas funciones, aunque son positivas, no son esenciales ni aumentan la productividad de la empresa. El metaverso inmersivo puede ir mucho más allá, y hacia aquí es donde se dirige este proyecto, hacia un producto de innovación tecnológica que sirva de vehículo para mejorar el mundo real.

Uniendo los conceptos de *digital twin* y metaverso industrial, el valor que hay detrás de la ciencia de los datos es complicado, pero el DT es capaz de captar y traducir la complejidad a una interfaz visual, representada como un mundo virtual réplica del existente, respondiendo dinámicamente a la información en tiempo real. Esto abre las puertas a unas potentes capacidades de simulación más sencillas que nunca.

## 2.2. Antecedentes y referentes

A pesar de que el uso del metaverso para controlar las oficinas a través de DT es una tecnología innovadora que todavía no se ha extendido, sí que hay algunos ejemplos relevantes sobre los cuales inspirarse. Si bien es cierto, la mayoría de ellos son del ámbito de la automovilística y más dirigidos a las fábricas y la cadena de montaje, no tanto hacia las oficinas, las lecciones de simulación que se pueden extraer de ellos son interesantes.

### 2.2.1. Colaboración BWM Group y NVIDIA

En 2021, y siguiendo con su ya conocida alianza, NVIDIA y BMW colaboraron para crear un *digital twin* de demostración para una planta de producción. El DT de la planta servía como representación de su homologo físico, creado mediante la recolección e interpretación de datos y algoritmos de IA. Esto le permitía a BMW simular y evaluar varios escenarios de fabricación dentro de un entorno virtual antes de implementarlo en el mundo real, cosa que reducía el riego de tiempos de inactividad y optimizaba la producción.

Esta colaboración subraya el impacto transformador que la tecnología de los DT puede tener, cuando se junta con el metaverso, en el rendimiento general de las instalaciones de trabajo.

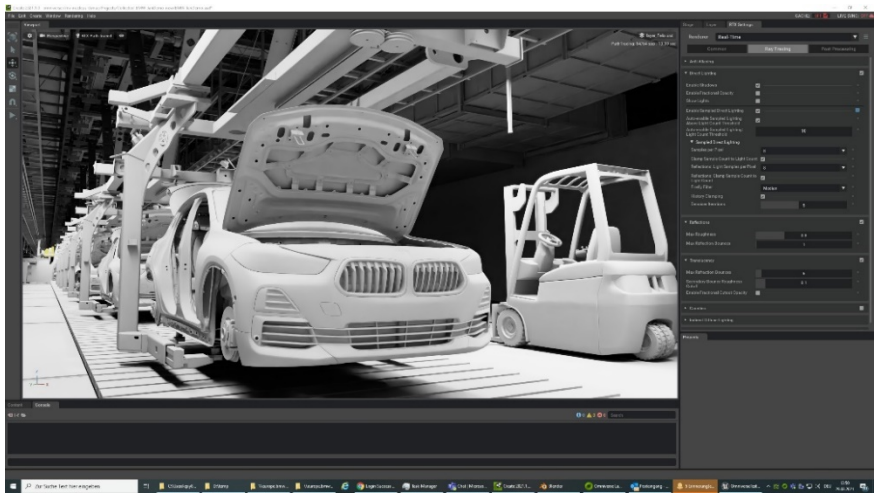

Figura 2.1. Muestra del *digital twin* en *Omniverse*. Fuente: BMW Group, 2021 [10].

### 2.2.2. Siemens y su construcción de edificios optimizada

Otro uso de los gemelos digitales aplicados al control de trabajo es el de Siemens. Esta empresa opta por esta tecnología para optimizar la eficiencia energética de sus edificios de oficinas. La empresa hace uso de sensores para recuperar datos sobre el funcionamiento de la construcción, como la temperatura, la iluminación y la ocupación, y la introduce en su propio gemelo digital. Esto permite que la empresa pueda simular y probar diferentes estrategias para optimizar el uso energético, implementando en el DT los cambios antes de que se realicen en el edificio físico.

Además, hacen algo que se quiere implementar en este proyecto, ya que supervisan en tiempo real sus sistemas en los edificios, incluidos calefacción, aire acondicionado, iluminación y electricidad. Así pueden detectar los errores antes de que se traduzcan en grandes costes.

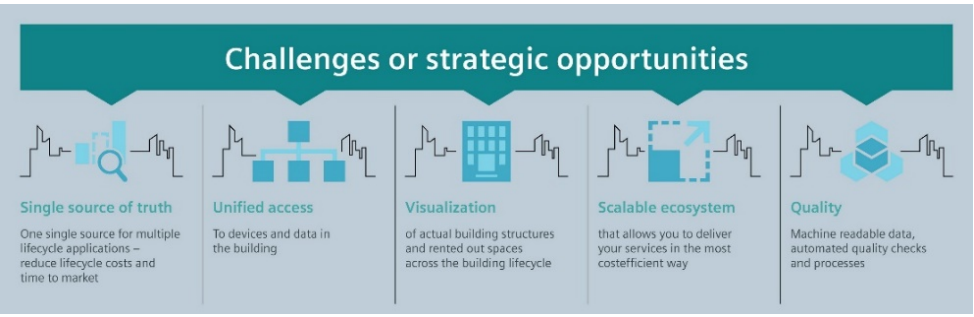


Figura 2.2. Retos/Estrategias de *digital twins* en oficinas. Fuente: Siemens, 2021 [11]

### 2.2.3. Colaboración Hyundai Motor y Unity Technologies

En 2022, Hyundai Motor y Unity Technologies anunciaron una colaboración para construir una fábrica en el metaverso, a la cual llamaron "metafábrica", con el objetivo de acelerar la innovación para las fábricas inteligentes. Desarrollaron un producto que aprovecha la plataforma de desarrollo 3d en tiempo real para simular las fábricas de coches. Esto permitía probar y optimizar los procesos de producción en un entorno simulado antes de implementarlo en la realidad. Con ello, pretendían impulsar la eficiencia y flexibilidad en la fabricación de automóviles, así como impulsar la innovación del sector.


Figura 2.3. Arquitectura de la "metafábrica". Fuente: Hyundai News, 2022 [1].

### 2.2.4. Conclusiones sobre antecedentes y referentes

A pesar de que hay tentativas sobre el control del espacio de trabajo haciendo uso del metaverso y los *digital twins*, es clara la falta de información y la falta de aplicación de estos en el trabajo de oficina de sus empleados.

Además, todos los ejemplos cuentan con la captación de datos y la aplicación de estos, pero ninguno de ellos expresa la bidireccionalidad de las decisiones tomadas en el mundo virtual y como estas afectan al entorno en tiempo real. Es decir, en ninguno de los ejemplos se utiliza el metaverso para afectar a los elementos físicos reales.

## 2.3. Análisis de tecnologías

Para realizar una propuesta sólida y clara del proyecto, es necesario hacer un análisis de tecnologías desde cuatro puntos de vista clave:

- Con qué herramientas crear el DT.
- Qué dispositivos utilizar para la recolección de los datos.
- Con qué herramientas crear el mundo virtual.
- Qué tecnologías utilizar para conectar los tres componentes en tiempo real.

### 2.3.1. Análisis de las herramientas para crear *digital twins*

El gemelo digital, de entre todas las opciones, se quiere crear en una plataforma *cloud* para así poder tener un intercambio de datos en tiempo real. A continuación, se comparan algunas de las plataformas basadas en la nube más importantes del mercado:

| **Funcionalidad** | **AWS** | **Microsft Azure** | **Google Cloud** | **IBM Cloud** |
|---|---|---|---|---|
| Herramientas para crear DT | AWS IoT Core, AWS IoT Greengrass | Azure IoT Hub, Azure Digital Twin | Google Cloud IoT, Google Cloud IoT Edge | IBM Watson IoT, IBM Digital Twin Exchange |
| Protocolo de comunicación | MQTT, HTTPS | MQTT, WebSockets, HTTPS, AMQP | MQTT, HTTPS | MQTT, HTTPS |
| Preparación y procesado de datos | Sí | Sí | Sí | Sí |
| Soporte 24/7 | No | Sí | Sí | No |
| Ofrece API | No | Sí | Sí | No |
| Precio | 0,116$ / MB | 10$ IoT unit / mes | 0,00045$ / MB | 0,001$ / MB |
| Visualización 3D | Sí | Sí | No | No |

Tabla 2.1. Comparativa de herramientas para crear DT [12].

Como Azure permite la visualización 3D y una API completa y documentada, se ha decido escoger esta herramienta. Además, ofrece una amplia gama de recursos fáciles de integrar entre ellos. Esto favorece la asignación de entidades, la creación de funciones de acceso a los recursos y permite la interacción sin autenticación. Además, la empresa que patrocina el proyecto es parte de Microsoft, cosa que favorece la elección de esta herramienta.

### 2.3.2. Análisis de los dispositivos IoT necesarios

Si se quiere crear y controlar la oficina a través del mundo virtual y el gemelo digital, estos son los dispositivos IoT imprescindibles con los que se debería contar:

- Iluminación: luces que puedan controlarse a través de alguna aplicación o reglas de automatización para optimizar así el consumo energético y los niveles de luz.
- Control de temperatura: termostatos que permitan programarse para ajustarse automáticamente a la temperatura que corresponda en función de la ocupación, la temperatura del exterior y otros factores que favorezcan el confort.
- Dispositivos de salud y bienestar: monitores para la calidad del aire y otros rastreadores de bienestar que permitan promover la comodidad de los empleados.
- Ocupación: sensores de movimiento para supervisar los espacios de trabajo.
- Enchufes: enchufes que permitan controlarse a distancia para encender o apagar dispositivos, supervisar su uso y reducir el gasto eléctrico.

Para el sistema de iluminación, se ha tomado la decisión de utilizar dos tipos de dispositivos finales, unas luces pequeñas y otras grandes. Se utilizarán para advertir a la gente de la sala sobre las condiciones del espacio, es decir, si factores como la humedad, el nivel de CO2, entre otros, son los correctos y la sala es usable. Básicamente, se usará en formato de alarma visual. Los dispositivos escogidos ya estaban implementados en la oficina de Barcelona de Avanade, y son las lámparas de mesa Philips Hue. Hay que tener en cuenta que es necesario disponer del Hue Bridge para poder hacer de puente entre las luces y todo el resto del sistema.

En cuanto al sistema del control de temperatura se han seleccionado el kit de desarrollo IoT MXChip AZ3166, una solución inteligente compatible con Arduino que permite recoger datos de humedad, temperatura, presión, fuerza magnética, aceleración y orientación. Es un hardware polivalente de fácil integración con Azure del cual ya se dispone en la oficina.

En lo referente a los dispositivos de salud y bienestar se ha decidido utilizar Netatmo Weather Station, una estación meteorológica inteligente que permite hacer seguimiento de los valores de temperatura, humedad, calidad de aire, avisos de ventilación y sonómetro. Se ha escogido este dispositivo dado que estaba disponible en la oficina, tiene una buena relación calidad/precio y ofrece una fácil integración con Azure.

Con relación al sistema de enchufes inteligentes, se han seleccionado los dispositivos Aeotec Smart Switch 7, un dispositivo orientado a controlar el consumo de electricidad de los dispositivos para mejorar el ahorro energético. Es un dispositivo pequeño y seguro muy bien valorado en base a su precio. Se utilizarán, en primera instancia, para conectar dos tótems de luz Newgarden Fity que son dispositivos no inteligentes de los cuales se dispone en la oficina para que cumplan la misma función que las luces Hue, pero en espacios más grandes y abiertos.

Respecto al sistema de sensores, se valorará la opción de integrarlos una vez estén los anteriores sistemas en funcionamiento. En caso de acabar implementándolos, en base a su facilidad de integración con Azure y su calidad/precio, se han escogido los sensores de movimiento Thirdreality Zigbee.

### 2.3.3. Análisis de las herramientas para crear mundos virtuales

Una vez decidido que el gemelo digital se va a desarrollar en Azure Digital Twins, es necesario ver que herramientas son compatibles para crear un entorno 3D que pueda conectar con él. Como se quiere que haya una posibilidad de acción bidireccional, no solo de visualización, se descartan los softwares de modelado de estructuras, tipo Autodesk Revit, los sistemas de automatización industrial, tipo *Siemens NX*, y las plataformas de IoT, tipo la que ofrece Amazon Web Services.

Por lo tanto, la opción que quedaría es la de optar por un motor de juegos, el cual facilita el trabajo de crear entornos interactivos, permitiendo simulaciones físicas, animaciones y visualización de datos en tiempo real, entre otras muchas cosas.

A la hora de decidir que motor escoger para conectar el mundo virtual con el *digital twin*, la decisión queda entre Unreal Engine o Unity:

- Unity es un motor muy popular y documentado, valido para crear mundos 2D y 3D. Es multiplataforma y tiene una gran comunidad detrás, lo que hace sencilla la tarea de buscar información. También incluye muchas herramientas y *plugins* para conectar con Azure Digital Twin.

- Unreal Engine es el otro motor de juegos más famoso. Este se especializa en la creación de mundo 3D con calidad gráfica superior y tiene buen soporte de VR. También incluye herramientas y *plugins* para conectar con Azure Digital Twin.

Entonces, viendo que ambas opciones son válidas e igual de correctas, la decisión se tomará en base a las características del proyecto, Por un lado, Unity ofrece mucha más información y comunidad que Unreal. Por otro lado, al estar modelando únicamente la oficina, la potencia grafica de Unreal no es necesaria. Finalmente, dado que la parte responsable del proyecto es más competente en Unity, se ha determinado que este software será la opción seleccionada.

### 2.3.4. Análisis de las tecnologías para conectar DT y mundo virtual

Dado que ya se ha establecido que se utilizará Azure Digital Twin y Unity, el abanico de tecnologías disponible para conectarlos es limitado. Básicamente, la opción documentada de comunicación se hace actualizando los sensores de Azure Digital Twin a través de los eventos de IoT Hub y se sincronizan con Unity mediante los WebSockets de SignalR.

Para conseguir que los sensores se actualicen dentro de Azure Digital Twin, es necesario que estén conectados a un punto de servicio que reciba los datos. Para ello se puede usar Azure IoT Hub, un servicio hospedado en el *cloud* que actúa como centro de comunicación entre el *digital twin* y los dispositivos conectados.

Después, se ha de crear una nueva suscripción a Event Grid, la herramienta *event broker* de Azure que permite la comunicación asíncrona entre aplicaciones. Esta tendrá que activarse cuando el DT reciba nuevos datos de los dispositivos inteligentes. Esto asegurará que los datos se propaguen por el DT en tiempo real lanzando una Azure Function.

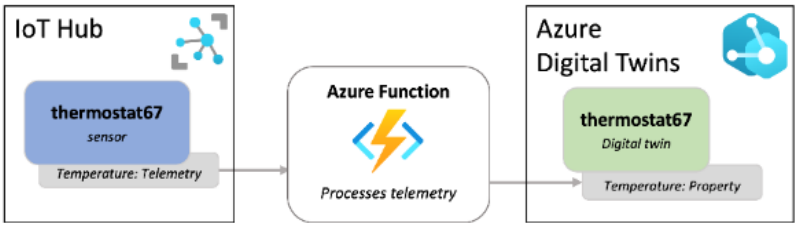


Figura 2.4. Ejemplo de evento desde IoT Hub a Azure DT. Fuente: Microsoft Learn [13]

Para que la conexión sea bidireccional, se deberá usar Azure Stream Analytics, una plataforma de procesamiento de eventos complejos y análisis en tiempo real que ofrece Azure para procesar y analizar volúmenes de datos de *streaming* de diferentes orígenes. Esta herramienta se encargará de recibir todas las actualizaciones de los dispositivos inteligentes, procesar los datos, y enviárselos al gemelo digital a través de una Azure Function.

Para conseguir la sincronización en tiempo real con el mundo virtual, es necesario utilizar SignalR, una librería ASP.NET que habilita precisamente esto. Una forma de implementar esto es utilizando el protocolo de comunicación de WebSockets que incluye. Este protocolo permite la comunicación bidireccional en tiempo real entre un cliente y un servidor a través de una única conexión de larga duración. Esto permite obviar el modelo tradicional de comunicación web basado en petición-respuesta.

Al conectarse al servidor SignalR, el cliente establecerá una conexión WebSocket si es posible. Esto permite que puedan intercambiar mensajes. El servidor actúa como centro que puede transmitir mensajes a varios clientes a la vez o enviar mensajes de forma individual.

Usando desde Unity la biblioteca de SignalR Client, se puede utilizar la API de Azure Digital Twin para que se suscriba a los cambios en el modelo y recibir las actualizaciones en tiempo real. Para lanzar estos cambios en el modelo, es necesario que el DT, mediante un recurso de Event Grid, invoque una Azure Function que conecte con el *hub* de SignalR para transmitir las actualizaciones. Unity será quien gestione las actualizaciones entrantes y las utilizará para la representación visual del *digital twin*. Para que la conexión sea bidireccional, el DT deberá tener una API que haga de intermediaria entre ambos servicios. Con esto se conseguiría tener el gemelo digital en Azure Digital Twin y el mundo virtual en Unity perfectamente sincronizados en tiempo real.

A continuación, se muestra un esquema para clarificar toda la explicación anterior, con las interacciones entre los diferentes recursos mencionados:

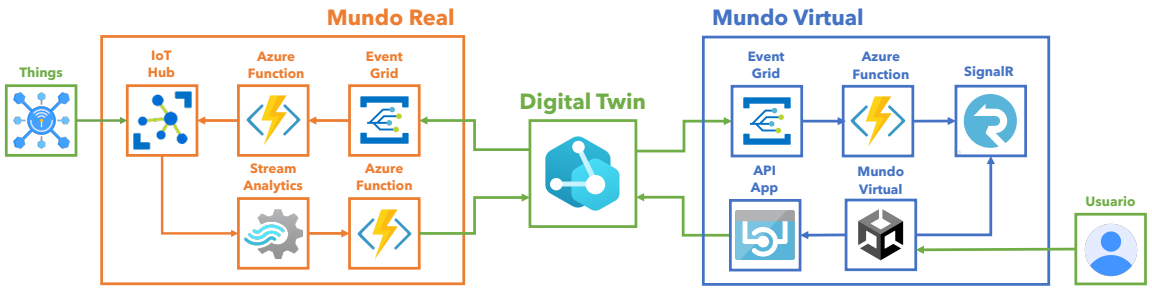


Figura 2.5. Esquema donde se muestra la arquitectura. Fuente: Elaboración propia

## 2.4. Necesidades de información

Para garantizar que se complete el proyecto con garantías de éxito, es necesario contar con la siguiente información:

- Contar con la API de Azure Digital Twins y que esté bien documentada.
- Contar con la API de los diferentes dispositivos inteligentes conectados.
- Contar con la documentación de cómo aplicar SignalR en Unity.
- Contar con la documentación de Azure Grid y Functions relacionada con *digital twin*.
- Contar con la documentación del lenguaje de programación C#.
- Contar con información de buenas prácticas para crear modelos 3D y texturizaciones.

Después de hacer una investigación suficientemente extensa se puede afirmar que todas las necesidades están cubiertas y son de acceso libre a través de documentaciones oficiales y foros actualizados y activos.

# 3. Objetivos y alcance

## 3.1. Objetivos

El primer y principal objetivo del proyecto es conseguir que una interacción dentro de un mundo virtual sobre un objeto genere una reacción y esta se refleje en el objeto del espacio real. Este producto permitirá:

- Crear un modelo digital de los objetos físicos, modelándolos y mapeándolos a sus contrapartes digitales.
- Conectar y compartir información entre un DT y un mundo virtual.
- Integrar el modelo de DT con una plataforma de mundo virtual que permita controlar y supervisar en tiempo real el entorno de la oficina, incluidas iluminación, temperatura y otras instalaciones.

En la medida de lo posible también se tendrán en cuenta los objetivos secundarios enumerados a continuación:

- Investigar y potenciar la experimentación en la interacción de DT con mundos virtuales y el análisis de datos en tiempo real.
- Explorar las implicaciones del uso de un mundo virtual para el trabajo a distancia, y las implicaciones que esto tiene en los espacios de oficina existentes.
- Investigar el potencial uso de entornos virtuales para mejorar la productividad y colaboración de los empleados. Esto podría incluir funciones como reuniones virtuales, herramientas de colaboración a distancia y seguimiento de las tareas en tiempo real.
- Investigar la utilidad y el retorno de diferentes dispositivos IoT en el ámbito de trabajo para proponer posibles ampliaciones.
- Mejorar la eficiencia energética de la oficina supervisando el control y uso de la electricidad en tiempo real.

## 3.2. Alcance

El desarrollo de la aplicación se hará utilizando Azure Digital Twins para la creación del DT y Unity para la creación del mundo virtual. Se creará el modelo 3D de la oficina de Avanade Barcelona desde cero usando Autodesk 3D Studio Max para el modelado y Adobe Substance Painter para la texturización. El almacenamiento y procesado de los datos se hará en Azure.

La interfaz del mundo virtual se hará altamente usable y clara para que cualquier persona pueda adaptarse de forma sencilla a las interacciones.

# 4. Metodología de trabajo

El desarrollo del proyecto se va a realizar siguiendo la metodología Scrum [14]. El trabajo se dividirá en diez iteraciones de dos semanas. Cada *sprint* incluirá todas las ceremonias que componen la metodología:

- ***Planning*** al inicio de cada *sprint*, donde se define el alcance del sprint y las tareas de cada PBI.

- ***Daily*** todos los días, donde el equipo se reúne para informar de los avances, exponer los objetivos del día y abordar obstáculos que puedan haberse producido.

- ***Refinement*** una vez por *sprint*, donde se valora el esfuerzo que supone cada PBI del proyecto y se le asigna un valor.

- ***Review*** **y** ***retrospective*** el último día de *sprint*, donde se muestra y evalúa el trabajo de la iteración para posteriormente recopilar comentarios y poder realizar los ajustes y mejoras necesarios en futuras iteraciones.

Se utiliza la metodología Scrum porque este proyecto se desarrollará de forma individual por la alumna, pero a la vez formará parte del equipo multidisciplinar de Innovación de Avanade Iberia, quienes aplican esta metodología en su trabajo del día a día.

La organización de las tareas del proyecto se va a realizar siguiendo la metodología Kanban, una herramienta de gestión de proyectos visual, iterativa e incremental diseñada para trabajar de forma más eficiente. Gracias a la herramienta Azure DevOps, se hará una división de los diferentes PBIs del proyecto en *sprints* y permitirá hacer un seguimiento del estado de las tareas de forma pública y visible para todos los integrantes del equipo. Las tareas se dividirán en tres fases: *To Do, In Progress* y *Done*. También, a cada una de las tareas, al igual que a los PBIs, se les asignará un valor que represente el esfuerzo que supone su implementación.

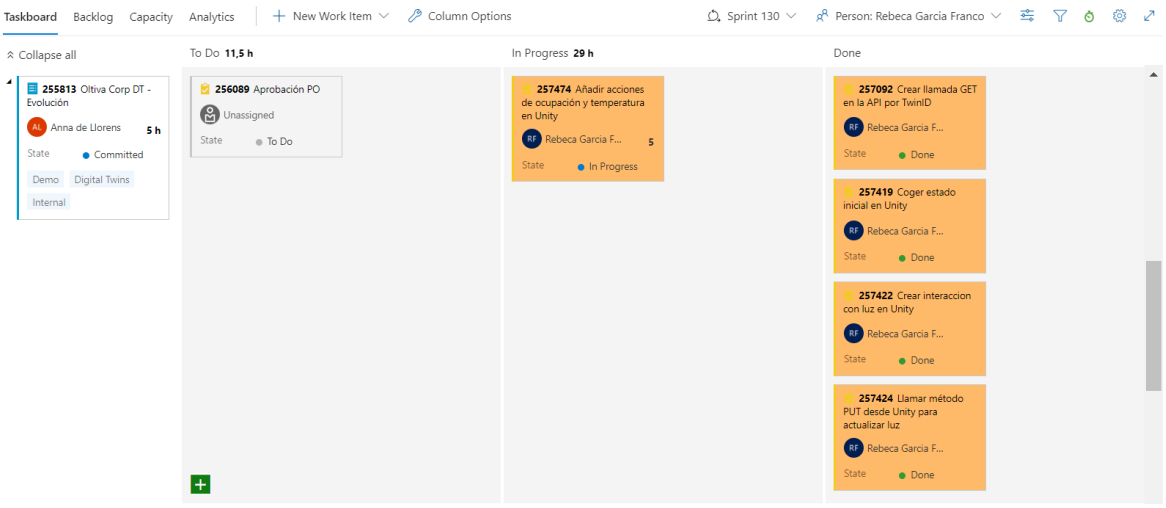


Figura 4.1. Ejemplo de tablero de este proyecte. Fuente: Azure Boards [15]

Todos los componentes del proyecto se implementarán siguiendo la metodología TDD (*Test-Driven Design*), una técnica de desarrollo de software basada en escribir las pruebas de la funcionalidad antes de empezar a escribir el código para implementarla. Esto garantizará que el código quede bien estructurado además de que sea fiable y fácil de mantener.

# 5. Análisis de los requisitos

Este apartado pretende hacer una investigación para determinar aquellos requisitos que permitan que el resultado del proyecto pueda controlarse y desarrollarse eficazmente.

### 5.1.1. Requisitos funcionales

La siguiente lista muestra las especificaciones técnicas que describen el comportamiento y las capacidades del sistema que limitan el alcance del proyecto:

- Permitir la gestión de datos de múltiples fuentes: el sistema deberá ser capaz de recopilar datos de diferentes fuentes externas, dispositivos IoT, para crear y rellenar de datos al DT.

- Facilitar la alta capacidad de agregación y transmisión de los datos: los datos tienen que procesarse en función de las características especificadas y transmitirse al mundo digital para que se puedan visualizar.

- Crear el *digital twin*: el sistema debe ser capaz de crear un DT del entorno físico de una oficina, incluyendo características estáticas y dinámicas.

- Visualizar datos de forma esquemática: el sistema debe proporcionar herramientas de visualización que permitan a los usuarios observar los diferentes comportamientos de forma significativa.

- Permitir la interoperabilidad: el proyecto debe ser capaz de integrarse con un entorno de mundo virtual creado de cero, permitiendo a los usuarios interactuar con el *digital twin* a través de él.

- Gestionar en tiempo real: lo que permitirá supervisar, analizar el rendimiento y transmitir la información del *digital twin* y del mundo virtual de forma que simule las interacciones de forma prácticamente instantánea.

- Habilitar las interacciones: hacer una exploración de la tecnología con la que se cree el mundo virtual para crear una experiencia inmersiva e interactiva en primera persona.

### 5.1.2. Requisitos no funcionales

Por lo concierne a aquellas funciones específicas de rendimiento y calidad del sistema, esta es la lista que los engloba:

- Favorecer la escalabilidad: el sistema debe ser resiliente a los cambios y permitir la conexión de nuevas fuentes de datos sin necesidad de rehacerlo, adaptándose a las nuevas necesidades de la oficina según avance el tiempo.

- Asegurar un buen rendimiento: al ser un sistema en tiempo real, es clave que el tiempo de respuesta sea rápido y la latencia sea baja.

- Afianzar la seguridad: se debe contar con protocolos de seguridad que garanticen que no se pueda acceder y manipular los datos obtenidos de forma no autorizada.

- Crear una infraestructura en el *cloud* segura y fiable: escoger un servidor robusto que permita manejar grandes cantidades de datos de forma confidencial, estable y con garantías de seguridad para evitar que agentes maliciosos accedan. Es este caso es especialmente relevante esto al ser un proyecto de empresa.

- Facilitar el uso: para extender la adopción y el uso generalizado, es necesario que el sistema sea accesible y los usuarios comprendan rápido su funcionamiento, reduciendo así el esfuerzo y aumentando la satisfacción, proporcionando una experiencia mejor.

- Documentar el proceso y el producto: es necesario que todo el proceso quede documentado para que pueda ser replicado y transmitido.

### 5.1.3. Requisitos tecnológicos y de infraestructura

Referente a la tecnología utilizada para desarrollar el sistema, estos son los requisitos específicos mínimos:

- Windows 10 u 11 en su versión de 64 bits.

- Suscripción de Azure con los recursos especificados.

- Controladores gráficos actualizados.

- 10GB de espacio en disco.
- Unity 2018.3 o posterior.
- Buena conexión a internet, estable y de baja latencia.

Referente a la tecnología necesaria para ejecutar el resultado final, se componen por:

- Windows 10 u 11 en su versión de 64 bits.
- Cuenta de Azure con los recursos especificados ya creados, implementados y en funcionamiento.
- *Build* de Unity.
- Buena conexión a internet, estable y de baja latencia.

# 6. Desarrollo

En este apartado se explican los resultados de cada una de las activades explicitadas dentro del apartado de *Planificación*. Se adjuntan todos los avances y las decisiones tomadas en cada uno de los *sprints*, siguiendo la metodología explicada en el apartado *Metodología*.

Todo el desarrollo está enfocado a los cuatro grandes hitos que quiere alcanzar el proyecto. Por ello, las diferentes actividades especificadas en el apartado de *Planificación Inicial* se engloban siguiendo la distribución siguiente:

1. Creación de una réplica digital de una oficina (actividad 1).
2. Reflejo de cambios del DT en un mundo virtual (actividad 2, 3 y 4).
3. Reflejo de cambios del mundo virtual en el mundo físico (actividad 5).

Cabe destacar, antes de profundizar en el proceso de desarrollo de la solución, que al ser un trabajo en colaboración con la empresa Avanade, existo un acuerdo de confidencialidad entre la empresa y la alumna encargada de realizar el desarrollo. Es por ello por lo que, a pesar de que se va a explicar todo el proceso de forma exhaustiva, no se van a mostrar fragmentos de código específicos del proyecto.

## 6.1. Contexto técnico inicial del proyecto

Antes de entrar en los detalles del desarrollo, es importante aclarar el punto de partida del proyecto. La empresa asociada para este proyecto, Avanade, ya contaban con un gemelo digital, llamado IntelligentSpaces, que era una solución anterior de gestión de espacios de trabajo basada en tecnologías de Microsoft. El producto se centraba en dos casos de uso principales:

- Confort térmico de las salas de reuniones: garantizar que el usuario experimenta un ambiente térmico confortable cuando empieza una reunión.
- Gestión del $CO_2$: medir el nivel de dióxido de carbono para gestionar la calidad del aire.

Para recopilar datos ambientales de los diferentes espacios de trabajo, se utilizaron los dispositivos MXChip AZ3166 y Netatmo para recoger mediciones interiores de temperatura y $CO_2$ respectivamente. Además, se necesitaban datos meteorológicos de las ciudades donde se realizaba la gestión, así que para ello crearon una Logic App que integraba el producto con la API OpenWeather. Posteriormente, todos estos datos se persistieron en una base de datos Azure y se insertaron en el gemelo digital. También detectaban el estado de las luces para poder utilizarlas en formato de alarma visual en caso de que los valores ambientales no fuesen correctos. Este producto se desplegó en tres oficinas: Madrid, Barcelona y Lisboa.

A continuación, se adjunta un esquema del proyecto:

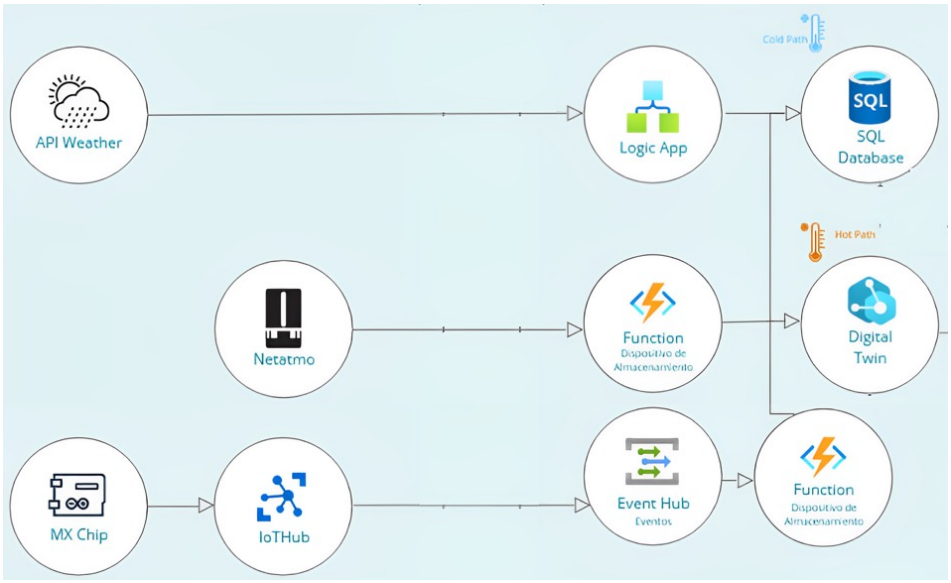


Figura 6.1. Esquema del DT actual de Avanade. Fuente: Avanade.

La parte del gemelo digital que implica a la oficina de Barcelona no es demasiado compleja. Tiene nodos que representan las diferentes salas, cada una con luces, temperatura, nivel de CO2 y humedad. Además, de cada una de las salas, extienden nodos que representan los espacios de trabajo, que cuentan con una propiedad que permite saber si están ocupados.

La telemetría previamente mencionada, se introduce en el gemelo digital a través de IoT Hub. Cada uno de los dispositivos reales está vinculado a un *device* creado dentro de esta herramienta. Cuando se producen datos de actualización, a través de la herramienta Stream Analytics, se unifican las diferentes entradas de los dispositivos y, a través de *queries*, se formatean y redirigen al output, en este caso una Azure Function, la cual conecta directamente con la instancia del gemelo digital y le introduce los datos.

El alcance de este proyecto sobre el desarrollo actual se centra únicamente en el uso de la parte del gemelo digital correspondiente a la oficina de Barcelona y la inserción de datos de telemetría en el mismo.

## 6.2. Diseño de la solución

Antes de entrar en los detalles del desarrollo, para poder tener una imagen general de cuál será la aplicación resultante en esta entrega, a continuación, se adjunta una muestra de la arquitectura actual según las necesidades y requisitos que presenta el sistema:

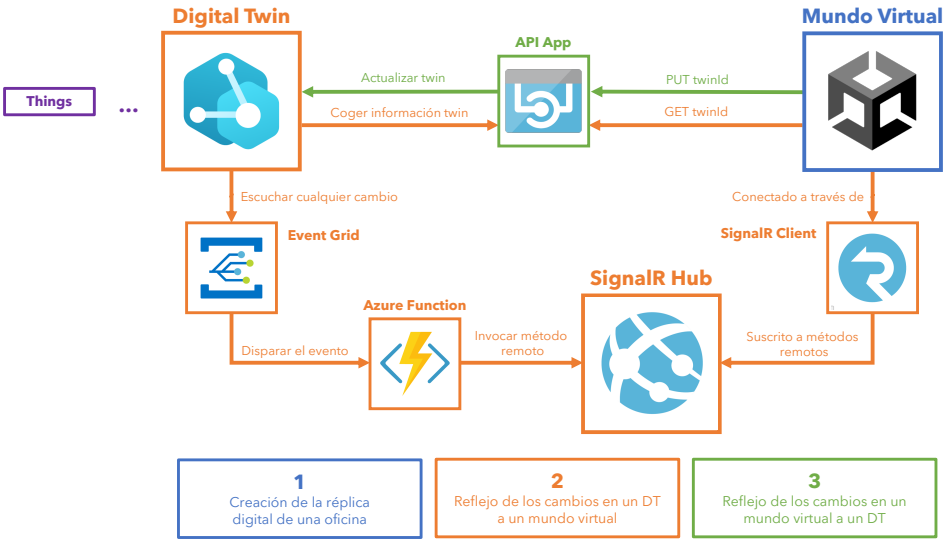


Figura 6.2. Arquitectura actual del proyecto. Fuente: Elaboración propia.

A lo largo de todo el ciclo de vida del proyecto, se ha garantizado la gestión y control de los cambios mediante repositorios Git para todos los componentes, así como la Azure Function, el SignalRHub, el Mundo Virtual en Unity y la API.

Además, dándole importancia a las medidas de seguridad, el almacenamiento de los *endpoints* y las contraseñas se ha centralizado en un recurso Azure KeyVault. Esto supone una mitigación del riesgo de exposición de información sensible, ya que las diferentes aplicaciones ya no tienen que incluir estos detalles dentro del código.

Por último, para facilitar el testeo de los recursos *cloud* después de desplegarlos, se ha utilizado Azure Insights, un servicio de monitorización y diagnostico que ofrece información en tiempo real de los diferentes recursos de Azure. Esto ha sido clave para verificar el comportamiento de los componentes y solucionar los diferentes problemas que han surgido a lo largo de todo el proceso.

## 6.3. Creación de la réplica digital de una oficina

En esta actividad, las tecnologías necesarias son: Unity, 3Ds Max y Substance Painter.

Previamente a la explicación de los pasos seguidos, cabe destacar que todos los modelos, incluidos los objetos, se han hecho desde cero, de forma propia y original. Dicho esto, para empezar con la creación de la réplica digital de la oficina, lo primero ha sido obtención de los planos y la toma de medidas de las paredes modificadas y todos aquellos objetos específicos relevantes que se querían incorporar al modelo final.

Una vez concretadas las medidas, se ha empezado el modelado en 3D, iniciado con el levantamiento de las paredes, uno de los pasos más importantes y críticos de todo este proceso. A continuación, se adjunta el resultado:

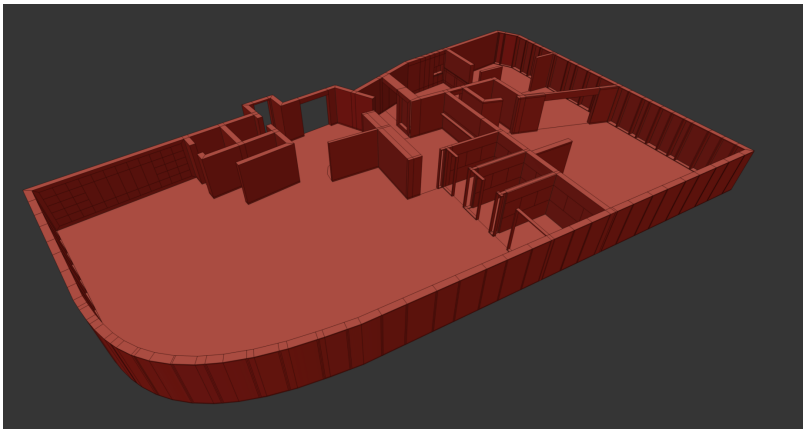

Figura 6.3. Modelado de las paredes y el suelo. Fuente: elaboración propia.

Lo siguiente, ha sido realizar una *mise en scène* de todos aquellos objetos característicos de la oficina y los dispositivos IoT necesarios para la posterior conexión con el *digital twin*. Una vez concretada esta, se ha procedido con el modelado de cada uno de ellos. Este es el conjunto de objetos finalizados:

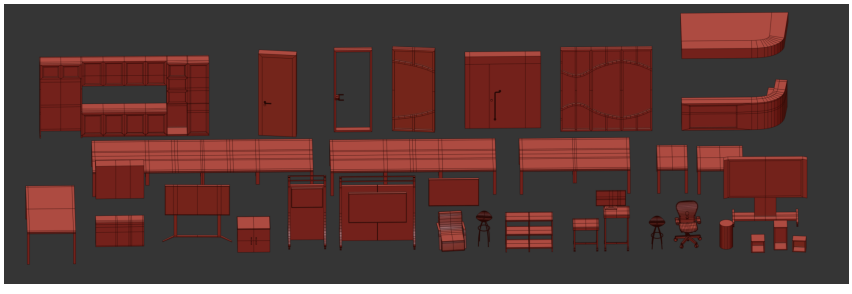

Figura 6.4. Modelado de los objetos. Fuente: elaboración propia.

Una vez el modelado ha estado listo, el siguiente paso ha sido hacer las UVs de todos los objetos, es decir, las coordenadas de textura 2D mapeadas a los vértices del modelo 3D, y empezar con la texturización, manteniéndose lo más fidedigna posible al mundo real, realizada a través de la herramienta Substance Painter. A continuación, se muestra como resulta el proceso de texturizado en uno de los objetos:

Figura 6.5. Texturización de una silla. Fuente: elaboración propia.

Una vez terminada esta parte, ya está terminado el mundo virtual, así que solo queda introducirlo en el motor de juegos donde se creará toda la lógica. Para ello, se ha creado un proyecto de Unity 3D en la versión 2021 LTS. Entonces, se ha creado el *prefab* de cada uno de los objetos, que son *GameObjects* preconfigurados que se crean en la escena y se almacenan en el proyecto. Esto permite que, una vez están todos colocados, en caso de tener que hacer algún retoque en modelos o texturas, todos los objetos de un mismo tipo se puedan cambiar de una sola vez, sin tener que modificarlos uno a uno. Cada *prefab* contiene el material del objeto compuesto por las texturas creadas y exportadas en el paso anterior. Después de colocar todos los objetos en el entorno, este es el resultado final:

Figura 6.6. Resultado del mundo virtual en Unity. Fuente: elaboración propia.

Para poder experimentar la sensación del usuario cuando más adelante pueda navegar por el espacio, se ha creado un controlador en primera persona muy simple únicamente para poder moverse actualmente. Esta es la vista que tendría el usuario en varios puntos del escenario:

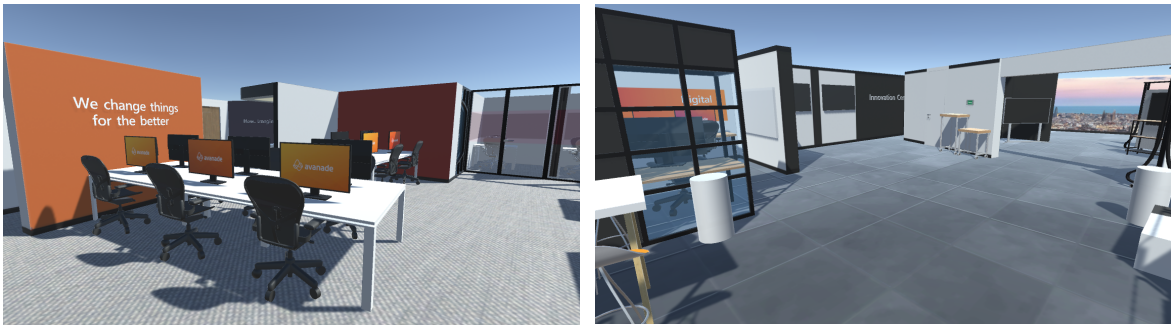


Figura 6.7. Muestras del mundo virtual en primera persona. Fuente: elaboración propia.

## 6.4. Reflejo de cambios del DT dentro de un mundo virtual

En esta actividad, las tecnologías necesarias son: Unity, SignalR y diferentes servicios de Azure, en concreto, Azure Web App, Azure Event Grid, Azure API app y Azure Functions.

### 6.4.1. Análisis previo sobre 3D Scene Studio

Este apartado se centra en la evaluación de la herramienta que ofrece Microsoft para la visualización de datos en tiempo real de un gemelo digital, conocida como Azure 3D Scene Studio [16]. Es apartado es necesario para entender sus funcionalidades y limitaciones, lo cual servirá de base a la hora de decidir qué y cómo implementar en el mundo virtual propio.

Azure 3D Scene Studio es una herramienta de Microsoft Azure diseñada para simplificar el despliegue de espacios 3D interactivos vinculados a gemelos digitales. En ella, se pueden crear entornos 3D inmersivos donde los usuarios pueden supervisar los datos operativos del gemelo digital al que estén vinculados en tiempo real. Y esta visualización se puede hacer desde el propio navegador web.

Antes de empezar el proceso de creación, para configurar el entorno, es necesario crear dos recursos en Azure, una instancia de Azure Digital Twin y otro de Azure Storage Container con un contenedor privado. El almacenamiento es necesario para guardar tanto el archivo 3D como el archivo de configuración, el cual contiene el mapeo entre el modelo y el gemelo digital y la lógica definida por el usuario.

Una vez está configurado el entorno, el primer paso necesario en la plataforma *low-code* es crear la escena, que es un entorno 3D con su propia lógica y referencias a una instancia de Digital Twin. Cabe destacar que se pueden tener múltiples escenas para un solo gemelo digital.

Para crearla, simplemente es necesario darle un nombre y tener el modelo 3D con las texturas encapsuladas exportado en formato GLB. Cabe destacar que el modelo no puede pesar más de 100MB para que la herramienta lo soporte. Como el archivo de este proyecto pesaba más, se han encapsulado las texturas en formato JPG al exportarlo. Así se ve la escena recién creada una vez dentro del editor:

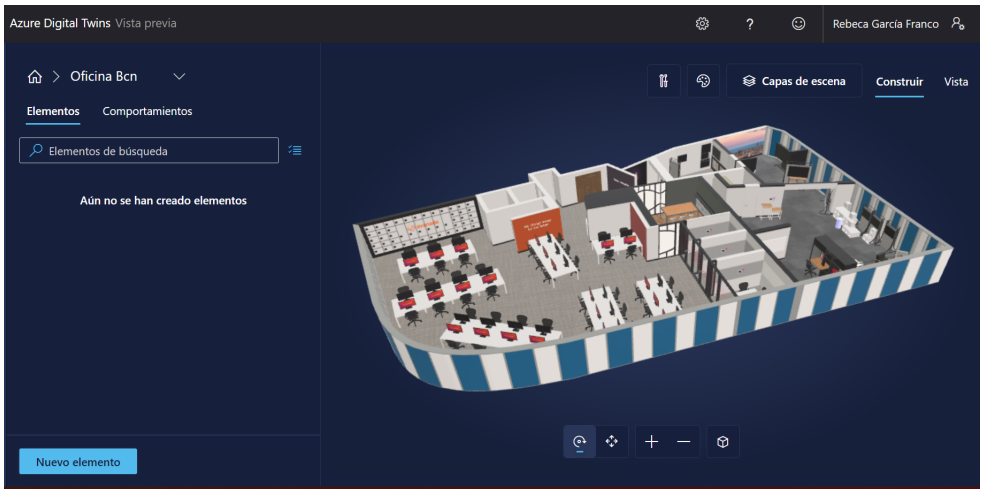


Figura 6.8. Escena de la oficina en 3D Scene Studio. Fuente: elaboración propia.

Una vez creado, es necesario generar los elementos y los comportamientos de la escena en el apartado llamado *builder* de la herramienta:

- Los elementos son asociaciones de mallas 3D con datos relevantes del gemelo digital. Para crearlos, se debe definir a que *twin* de la instancia se vincula, llamado *primary twin*, los objetos a los que afecta, un nombre y los comportamientos que deban producirse.

- Los comportamientos son descripciones de cómo deben mostrarse los elementos en la visualización. Existen dos tipos de comportamiento: las reglas visuales, que son superposiciones de colores o iconos en los elementos, y los widgets, que son una ventana emergente en la que se puede personalizar como se muestra la información deseada.

Una vez se tiene una comprensión clara de las posibilidades de la herramienta, el siguiente paso consisten en empezar a crear los comportamientos que van a gestionarse. La siguiente lista describe los diferentes comportamientos que se han creado:

- Alerta Temperatura: este comportamiento contiene una regla visual que ajusta el color de la pantalla del termostato en función de su valor. Además, incluye un widget que, al desplegarse, muestra el valor de la temperatura, la humedad y la calidad del aire en la sala.

- Luz Encendida: este comportamiento contiene una regla visual que señala el estado de la luz al establecer de color azul el objeto cuando está encendida.

- Sitio Ocupado: este comportamiento contiene una regla visual que señala el estado ocupado de un lugar de trabajo al establecer de color rojo la silla que corresponde.

Una vez están creados los diferentes comportamientos, es necesario asociarlos con los elementos que van a implementarlos. A continuación, se muestra la tabla que relaciona los elementos creados con el objeto donde se han creado y los diferentes comportamientos asociados:

| **Elementos** | **Objeto donde se implementa** | **Comportamientos asociados** |
|---|---|---|
| BIC – Temperatura<br>EduardPunset – Temperatura<br>JosefinaCastellvi – Temperatura<br>JosepCarreras – Temperatura | Pantalla del termostato de la sala | Alerta Temperatura |
| BIC – Totem 01<br>BIC – Totem 02 | Tótems de luz | Luz Encendida |
| BICStaff – 01<br>BICStaff – 02<br>BICStaff – 03<br>BICStaff – 04<br>BICStaff – 05 | Sillas de los puestos de trabajo | Sitio Ocupado |

Tabla 6.1. Elementos y comportamientos implementados en la escena.

Así se ve el resultado final en el apartado *viewer* de la herramienta, es decir, el visor en tiempo real de la información del *twin* según los elementos y comportamientos definidos:


Figura 6.9. Captura de la escena en funcionamiento. Fuente: elaboración propia.

Y así se vería el detalle al hacer clic sobre un objeto, es este caso, el termostato, el cual incluye un *widget*:

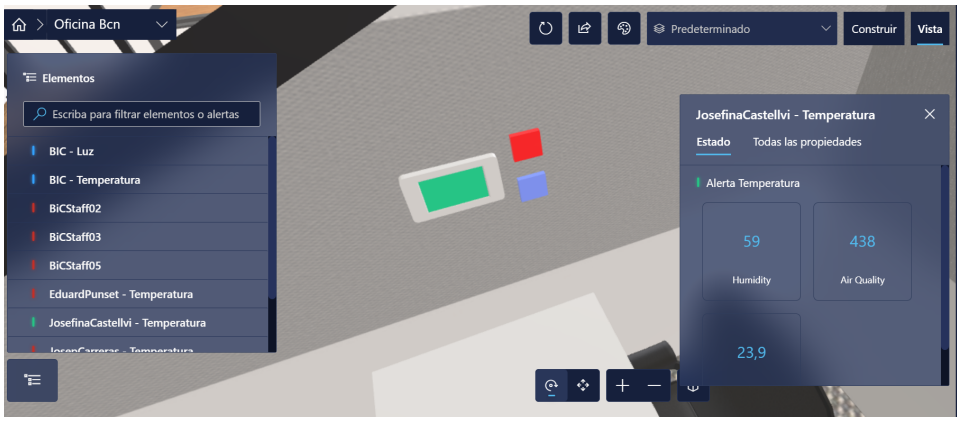


Figura 6.10. Captura del detalle de los termostatos. Fuente: elaboración propia.

Después de hacer esta implementación, se puede concluir que la herramienta aporta un visor sencillo y fácil de utilizar que ofrece una lectura de la información del gemelo digital en tiempo real de forma eficiente. El problema principal de esta herramienta es que no permite la comunicación de forma bidireccional. Es por eso por lo que se va a hacer una implementación propia siguiendo la utilidad que plantea esta herramienta, pero ofreciendo una visión en primera persona y la posibilidad de interactuar sobre los elementos para poder modificarlos.

Además, como hace uso de la API estándar de Azure Digital Twin, cuando se añaden muchos elementos y comportamientos, el sistema empieza a ralentizarse debido a la alta cantidad de llamadas que hace debido a su actualización regular de datos, que al tener menos de cien *twins* es cada diez segundos. Es decir, que el software se resiente mucho por el hecho de tener que estar llamando a la API cada escasos segundos para ver si se ha producido un cambio. Es por ello por lo que es interesante crear un sistema que sea proactivo, es decir, que sea él el que notifique cuando se produce un cambio en el gemelo digital y no que sea el software de visualización el que deba estar preguntando si se ha producido algún cambio. Y esto también va a implementarse en la solución propuesta.

### 6.4.2. Comunicación en tiempo real con entorno propio

Derivado de las conclusiones del apartado anterior y el mundo virtual ya creado, el desarrollo de este apartado se centra en crear la comunicación en tiempo real con el *digital twin*. Para ello, tal como se explica en el marco teórico, se ha utilizado SignalR, una librería de ASP.NET que simplifica la comunicación de este tipo proporcionando llamadas a procedimientos remotos des del servidor, llamado *hub*, a través de mensajes con el nombre y los parámetros del método que implementa el cliente [17]. Cabe destacar que, para que no de error de versiones, tanto el *hub* como los clientes que quieran utilizar SignalR hacen uso de versiones compatibles .NET Standard 2.0, que es el que implementa y permite utilizar Unity.

Por lo tanto, antes de entrar en la comunicación entre DT y mundo virtual, lo primero es crear el *hub*, que es donde se almacenan los métodos que los clientes conectados pueden llamar. También implementa los métodos a los que los clientes podrán suscribirse si están interesados. Básicamente es la clase que sirve como *pipeline* de alto nivel y gestiona la comunicación cliente-servidor.

Para implementarlo se tiene que crear un proyecto ASP.NET de tipo Core Web App en C# que utilice .NET 6.0. Se ha utilizado la versión 1.1.0 de ASP.NET Core SignalR. Después se ha creado el *hub* declarando una clase que hereda de Hub, que no obliga a implementar ningún método, pero aporta las bases para gestionar las conexiones, los grupos y los mensajes.

A partir de aquí, se tienen que añadir los métodos públicos a la clase para hacer que los clientes puedan llamarlos. Cabe destacar que, en caso de que el método sea asíncrono y necesite que el *hub* permanezca vivo, se tiene que utilizar el *await*, ya que cada llamada a un método del *hub* se ejecuta en una nueva instancia de este, por lo tanto, puede dar error en caso de que finalice antes el método que la sentencia asíncrona.

A continuación, se muestra un pequeño ejemplo a modo de plantilla sobre cómo crear un *hub* de un método:

```
using Microsoft.AspNetCore.SignalR;

namespace UnitySignalRHub
{
    public class UnityHub : Hub
    {
        public async Task SendAMessage (string user, string message)
        {
            await Clients.All.SendAsync("ReceiveAMessage", user, message);
        }

        // AÑADIR EL RESTO DE LOS MÉTODOS
    }
}
```

Figura 6.11. Plantilla de *hub* de SignalR. Fuente: elaboración propia.

Los métodos tienen que ser públicos y devolver una instancia de *Task*, que representa una operación asíncrona que no devuelve ningún valor. Más adelante se mostrará como los clientes pueden llamar de forma remota y recibir los resultados de estos métodos.

Después de esto solo queda configurar que archivo de programa para añadir el servicio de SignalR al contenedor de inyección de dependencias y mapear el *endpoint* del *hub* personalizado que se ha creado.

Por último, para tener este servidor disponible en todo momento, se ha creado un recurso en Azure de tipo Azure Web App, que permite fácilmente implementar y ejecutar aplicaciones web en la nube. Una vez publicado el código en la instancia a través de Visual Studio, los clientes podrán suscribirse al *hub* utilizando el *endpoint* de Azure y añadiendo la extensión que se haya mapeado en el archivo de programa.

Después de crear el *hub*, el siguiente paso es configurar un sistema para detectar eventos de actualización en la instancia de Digital Twin subida en Azure y comunicarlos al servidor SignalR. Esto requiere cuatro pasos:

1. Crear la instancia de Azure Event Grid.
2. Crear el *endpoint*.
3. Crear la *event route*.
4. Crear y lanzar la Azure Function.

Antes de nada, hay que destacar que para que este proceso pueda realizarse, el usuario debe ser propietario de los datos dentro de la instancia del DT.

Para empezar, se crea una instancia de Event Grid, que es un servicio de Azure que facilita el enrutamiento y la entrega de eventos a otros servicios. Este recurso recoge eventos del DT, y los suscriptores pueden escuchar la instancia para recibir los eventos a medida que llegan.

A continuación, se crea el *endpoint* dentro de Azure Digital Twin y la ruta que le enviará los eventos. Cuando el DT se actualice, utilizará la ruta para enviar información sobre los eventos de actualización al *endpoint*. Event Grid recogerá los eventos desde este y los pasará a la Azure Function para su posterior procesamiento. Tanto el *endpoint* como la ruta pueden ser creados en la sección *"Connected Outputs"* de la instancia Digital Twin.

Cabe destacar que se pueden establecer filtros dentro de la ruta para limitar el tipo de eventos que se transmitirán. En este caso, se requieren eventos de tipo actualización, por lo que se ha aplicado el siguiente filtro a la ruta: *type = 'Microsoft.DigitalTwins.Twin.Update'*.

Lo siguiente es crear la Azure Function que escuchará a Event Grid y recibirá los eventos para procesarlos y enviarlos al *hub* SignalR según la lógica necesaria. Para ello, primero se crea la instancia del recurso y luego se crea el proyecto dentro de Visual Studio. Después, se añaden las dependencias para recibir eventos de Event Grid. Una vez que está todo configurado, queda rellenar la lógica de la función y publícala.

Para la lógica de la función, simplemente cuando recibe la información del evento hace un *parse* a JSON. Después, crea la conexión con el *hub* de SignalR y la empieza. A continuación, distingue el tipo de dispositivo del que procede el evento y actúa de una forma u otra dependiendo de esto. De momento, están implementados los eventos de luz y de temperatura.

Para hacer que funcione la Azure Function, solo queda publicarla en la instancia creada anteriormente y conectarla a Event Grid creando una suscripción desde la sección "Event Subscriptions" dentro de las "Entities" de la instancia de Event Grid. Una vez completado, el *hub* SignalR comenzará a recibir eventos de actualización de DT.

Lo último que queda hacer es que el mundo virtual reciba los eventos que se lanzan cuando se produce una actualización en el DT. Para esto, Unity, como cliente, tiene que implementar el paquete SignalR Client, pero tiene un problema, que el motor no implementa esto de forma nativa. Entonces, para resolverlo, la solución ha sido crear un proyecto en Visual Studio de consola vacío e importar des del Nugget Packet Manager la librería. Des de ahí, en la carpeta del proyecto, se puede arrastrar el DLL directamente a una carpeta llamada *Plugins* dentro de los *assets* del proyecto de Unity.

Esto conlleva otro problema, las dependencias. Ahora el proyecto no compila porque falta dependencias que el cliente de SignalR necesita, por ello, según se marcan en el inspector y siguiendo el mismo proceso anterior, se van arrastrando todos los DLLs necesarios uno a uno a la carpeta de *Plugins* del proyecto hasta que compila y puede ejecutarse.

Una vez el proyecto ya tiene las dependencias del cliente de SignalR bien configuradas, solo queda crear el script necesario para que se detecten este tipo de mensajes y actuar en función del evento que llegue. Esta es una plantilla simple de los cuatro pasos necesarios para implementar el *hub* ejemplo que se ha mostrado al inicio de este apartado:

1. Crear la conexión cuando se ejecute la aplicación.
2. Suscribirse a los eventos que interesa recibir y definir cuál va a ser la lógica que se ejecutará cuando se reciban.
3. Empezar la conexión.
4. Parar la conexión cuando deje de ejecutarse la aplicación.

```
using System;
using Microsoft.AspNetCore.SignalR.Client;
using UnityEngine;

public class SignalRClient : MonoBehaviour
{
    [SerializeField] private string url;
    private HubConnection _connection;

    private async void Start()
    {
        _connection = new HubConnectionBuilder()
            .WithUrl(url)
            .Build();

        _connection.On<string, string>("ReceiveAMessage", (user, message) =>
        {
            // LÓGICA QUE SE EJECUTARÁ AL RECIVIR EL EVENTO
        });

        await _connection.StartAsync();

        Debug.Log("Connected...");
    }

    private async void OnApplicationQuit()
    {
        await _connection.StopAsync();
    }

}
```

Figura 6.12. Plantilla de cliente de SignalR desde Unity. Fuente: elaboración propia.

A partir de aquí solo queda replicar todo este proceso para identificar que eventos quiere recibir el mundo digital y como los quiere gestionar. Este es un resumen de los eventos que hay implementados actualmente en el Hub, como se lanzan desde la Azure Function, como se reciben en Unity y cuál es la lógica que se produce dentro del mundo virtual:

| Evento del Hub | Lógica que lanza el evento | Reflejo en Unity |
|---|---|---|
| SendMessage (string user, string message) | Clients.All.SendAsync ("ReceiveMessage", user, message); | Evento simple que se limita a imprimir por consola los dos *strings* que recibe: el nombre del usuario y lo que ha dicho. |
| LightChanges (string device, bool state) | Clients.All.SendAsync ("LightHasChanged", device, state); | Enciende o apaga la Point Light del tótem de luz indicado según el estado que le llega. |
| TemperatureChanges (string device, float temperature) | Clients.All.SendAsync ("TemperatureHasChanged", device, temperature); | Actualiza la temperatura del termostato indicado en el parámetro de texto que contiene y actualiza el color de su pantalla a azul si hace frio, verde si está bien o rojo si hace calor. |
| OccupationChanges(string device, bool occupied) | Clients.All.SendAsync (“OccupationChanged”, device, occupied) | En caso de que se ocupe un puesto de trabajo, enciende la pantalla del monitor y pone en rojo la silla del puesto. |

Tabla 6.2. Recopilación de los métodos disponibles en el SignalR Hub.

Para que pueda reflejarse en Unity el cambio de estado, se deberán crear tres scripts diferentes, uno para las luces, otro para los termostatos y otro para los puestos de trabajo. Todos comparten una estructura similar, que es la siguiente:

1. En el método *Start* obtienen las referencias a los subcomponentes que se ven afectados por el cambio que corresponda. Por ejemplo, en el caso de la ocupación, se obtiene la referencia a la silla y la pantalla de ese puesto de trabajo.

2. Implementación del método publico *{Light/Temperature/Occupation}Change* que tiene como parámetro el dispositivo y el estado. Este método es al cual se llama des del paso anterior para afectar a los objetos de una forma u otra. Por ejemplo, en el caso de la luz, si el estado fuese un false, se desactivaría el campo emisivo del material del objeto.

En la escena, los objetos deberán nombrarse igual que los nodos de la instancia del gemelo digital para que, cuando llegue una actualización, usando el método Find propio de Unity y el parámetro del dispositivo, se encuentre la referencia a los objetos afectados en la escena. Una vez obtenidas estas referencias, solo queda que se llame al método público del objeto afectado y este aplique la lógica según corresponda. Esto permitirá la comunicación en tiempo real des del gemelo digital hacia Unity, mostrándose asi los cambios según sucedan sin necesidad de llamar al SDK de Azure Digital Twin de forma regular como hacia 3D Scene Studio.

### 6.4.3. Conseguir el estado inicial

Antes de acabar el apartado, cabe destacar que, cada vez que se ejecute el proyecto de Unity, este debería obtener el estado en el que se encuentra el DT, porque si no haría falta que cada *twin* se actualizase para que en el mundo virtual se reflejase el verdadero estado del DT. Para ello, se ha creado una API que hace de intermediaria entre ambos, para así evitar desde aplicaciones de terceros acceder directamente con el SDK de Azure Digital Twins. El ella, se ha incluido un método GET que, al incluir el id de un *twin*, proporciona todo el estado actual, el estado de cada una de sus propiedades.

La API se trata de un proyecto .NET 7 posteriormente publicado en un recurso API App de Azure. A continuación, se adjunta un diagrama de clases y una explicación de la arquitectura utilizada:

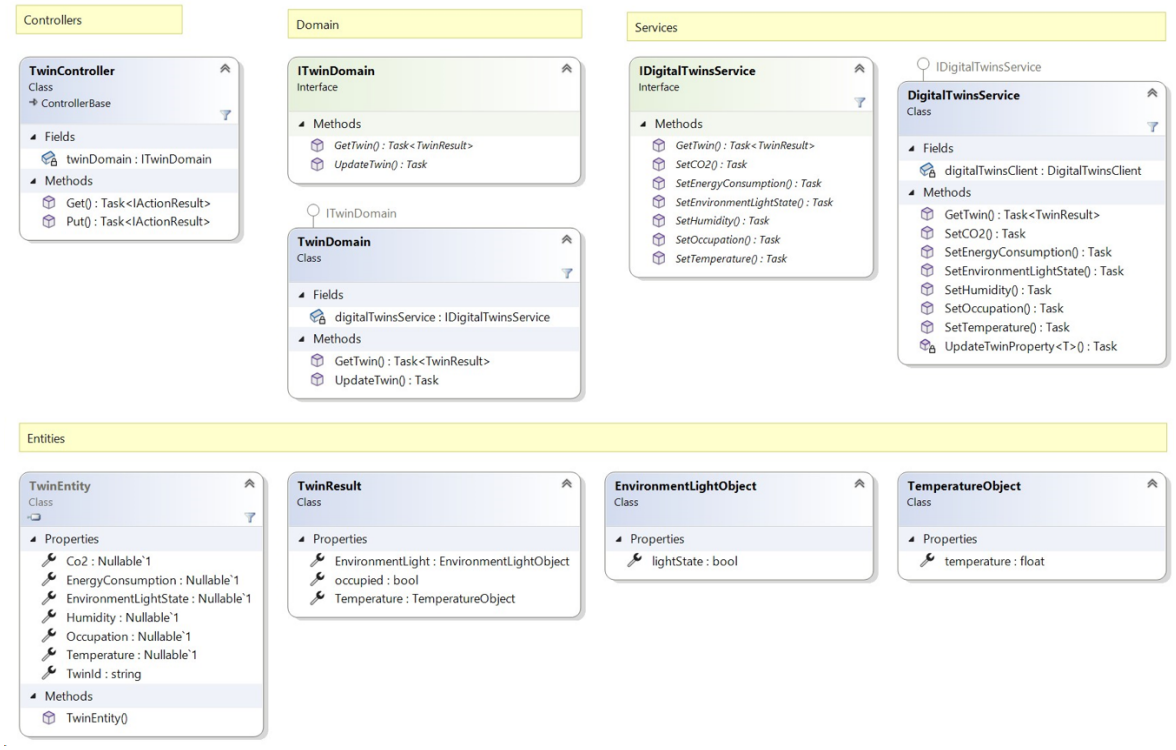


Figura 6.13. Diagrama de clases de la API. Fuente: elaboración propia.

El proyecto, en línea con la metodología que propone la empresa, sigue una estructura DDD (*Domain-Driven Design*). Esta metodología, aplicada a este caso en concreto, se basa en las tres capas que se explican a continuación:

- *Controllers*: Esta capa es la encargada de gestionar las peticiones HTTP y devolver las respuestas que correspondan. Es el punto de acceso para los clientes. En este caso, solo existe un método GET con la ruta *api/Twin/{twinId}* que, al ejecutarse, llama a un método del dominio llamado *GetTwin*, el cual incluye como parámetro el identificador especificado en la ruta. Al cliente se le devuelve un JSON que incluye toda la información del *twin* especificado en la petición en caso de que exista. Si el identificador no existe, al usuario le llegará un JSON en blanco.

- *Domain*: Esta capa es la parte fundamental de este tipo de arquitecturas. En ella se definen las reglas y comportamientos de la lógica de negocio. La capa de dominio sirve simplemente para llamar a la capa de servicio y devolver el resultado al controlador. En el caso, la capa contiene dos subapartados, los contratos y las implementaciones. Los contratos son las interfaces que definen el dominio y las implementaciones son precisamente las clases que implementan las interfaces previamente mencionadas. Esta API solo contienen una interfaz, llamada *ITwinDomain*, donde se define el método *GetTwin* que toma como parámetro el nombre del *twin* del que se desea la información y una implementación llamada *TwinDomain* que llama un método de los servicios llamado *GetTwin* que toma como parámetro el identificador que se había pasado como parámetro desde el controlador. Cuando le llega la respuesta del servicio, le pasa el JSON al controlador.

- *Services*: Esta capa es la responsable de implementar la lógica especifica de la interacción con el SDK de Azure Digital Twin. Al igual que la capa anterior, esta también se divide en contratos e implementaciones, pero incluye un tercer subapartado llamado entidades, donde se incluye la clase *TwinResult*, utilizada para representar el resultado que se obtiene al llamar al método GET del SDK. Básicamente, importando el paquete *Azure.DigitalTwins.Core* se utiliza la clase *DigitalTwinsClient* para implementar todas las operaciones que se ofrecen desde el controlador. En caso de hacer un GET, esta clase llama al método *GetDigitalTwinAsync* con el identificador que se le pasa por parámetro y devuelve el resultado “deserializado” en forma de *TwinResult*, una clase creada para únicamente considerar y devolver aquellas variables que son relevantes para este proyecto y se controlan en el gemelo digital, que son la temperatura, la luz y la ocupación.

Entonces, una vez creada la API, se ha creado un recurso de Azure llamado API App. Este recurso facilita el desarrollo, el alojamiento y la seguridad de los proyectos API REST, asegurando la total disponibilidad del recurso para la recuperación de datos del DT cuando sean necesarios.

Por temas de seguridad, se ha protegido la API utilizando la autorización OAuth 2.0 con *Azure Active Directory* (AAD), así las aplicaciones deberán adquirir y presentar un token válido para poder acceder a la API. Para ello, en primer lugar, se debe crear un *App Registration* para la API. Una vez creado, se deben configurar los permisos de aplicación para que pueda acceder el cliente dentro del apartado App Roles.

Para garantizar que el proceso de autorización de la API funcione sin problemas, se debe añadir en la instancia de API APP el registro como proveedor de identidad. Además, también se añade en este paso que, en caso de fallo por autorización, se lance un error HTTP 401-*Unauthorized*. Por último, la audiencia de la API debe editarse para incluir el identificador del registro.

Una vez la configuración de la API esta lista, es momento de configurar al cliente. Para ello, se debe crear otro registro de aplicación y concederle el rol que se ha creado antes en el registro de la aplicación. A partir de aquí, los clientes ya pueden solicitar el token, le cual permitirá una comunicación segura entre el cliente y la API, asegurando que se otorgue acceso autorizado.

Pero, antes de que el mundo virtual solicite el token, es necesario modificar el código de la API para que cuando entre una petición, solo se pueda completar si se incluye la autenticación. Solo hace falta que desde la clase *Program*, el punto de arranque del programa, se coja la información sensible del Key Vault sobre el registro de aplicación, se le indique que utilice autorización y autenticación y se añada en el constructor de la aplciacion. También es necesario incluir en el controlador la cabecera *[Authorize]*, que es lo que obliga a que, para que se complete la petición, la llamada incluya el token.

Por lo tanto, ahora solo queda que la aplicación de Unity obtenga un token de registro para utilizar la API y poder así realizar la llamada GET para obtener el estado inicial de los objetos.

Lo primero que se debe hacer es llamar al Key Vault desde Unity y coger los datos del *tenant* de Azure donde se encuentren los registros, el identificador de ambos y el secreto del registro de cliente. Después, como no se dispone de la librería Microsoft Authentication, se debe de hacer una llamada de tipo POST a la API de Microsoft Graph como la siguiente:

```
IEnumerator GetToken()
{
    string _resource = $"https://login.microsoftonline.com/{_tenantId}/oauth2/v2.0/token";

    WWWForm form = new WWWForm();
    form.AddField("grant_type", "client_credentials");
    form.AddField("client_id", _clientId);
    form.AddField("client_secret", _clientSecret);
    form.AddField("scope", _scope);

    UnityWebRequest request = UnityWebRequest.Post(_resource, form);
    request.SetRequestHeader("Content-Type", "application/x-www-form-urlencoded");

    yield return request.SendWebRequest();

    if (request.result == UnityWebRequest.Result.Success)
    {
        // GUARDAR TOKEN
    }
}
```

Figura 6.14. Estructura de petición para crear token en Unity. Fuente: elaboración propia.

Esta llamada, si se completa con éxito, devolverá una respuesta con un token de acceso que se persiste en una clase para "deserializar" el JSON llamada *TokenResponse*, que solo contiene un parámetro de tipo *string* llamado *access-token*. Todos los objetos del proyecto podrán acceder a él utilizando su método *getter*.

El problema es que el token tiene una hora de duración. Por eso, se ha creado una corrutina que vuelva a lanzar esta petición después de una hora, si el programa todavía sigue ejecutándose, para poder refrescarlo.

Ya teniendo el token, solo queda crear un script que permita que, teniendo un identificador de *twin*, llamar al método GET de la API para obtener toda la información de este. Para ello, se debe añadir un parámetro *SerializeField* tipo *string* llamado id donde se debe escribir el mismo nombre que el nodo de la instancia del gemelo digital a la que queramos acceder. Después, solo queda que, cuando ya se haya obtenido el token de acceso, crear una llamada de tipo GET que acceda a la uri de la API con la extensión /api/Twin/{id}, la cual se hace a través de una corrutina llamada *GetInitialState*. Se debe añadir la cabecera *Authorization* con valor *$"Bearer {token}"* para que funcione. Esto devolverá la estructura personalizada que se ha creado antes en la API, llamada *TwinResult*, la cual se "deserializa" en Unity, pudiendo saber cuál es el estado actual de ese *twin*.

Este script se ha implementado en todos los objetos que deben recibir cambios, cada uno con el identificador de nodo que le corresponda como parámetro.

Después de todo este proceso, cuando ya se tiene la información, se activan los scripts de control de temperatura, luz y ocupación en su método de actualización de estado con el valor que corresponda. Así, cada vez que se lance la aplicación, todos los objetos se actualizarán con el estado actual del *twin* en la instancia de gemelo digital de Azure en ese preciso momento.

## 6.5. Reflejo de cambios del mundo virtual en el mundo físico

En esta actividad, las tecnologías necesarias son: Unity y Azure API app principalmente.

El primer paso para reflejar las acciones del mundo virtual en el mundo físico es necesario detectar cuando se producen estas dentro de Unity. Para ello, se ha seguido una lógica similar en todos los casos, tanto para la temperatura, como para la ocupación y la luz.

Lo primero que es necesario es implementar un método en el script de control de movimiento del jugador que lance un haz, conocido como *Raycast*, cada vez que se pulsa el clic derecho del ratón. Esta funcionalidad propia de Unity permite lanzar un rayo desde un punto especificado para detectar el objeto que se interpone en la trayectoria. Una vez se devuelve el objeto detectado, se intentan invocar los scripts de temperatura, ocupación y luz. En caso de que no dispongan de ninguno, no se hará nada, pero si el objeto golpeado tiene alguno de los tres, se activará una de las siguientes lógicas:

- En el caso de que el objeto detectado sea una luz o un escritorio, se llamará al evento de actualización con la variable booleana que guarda el estado actual invertida.

- En el caso de que el objeto detectado sea el botón de subir (rojo) o bajar (azul) la temperatura del termostato se llamará al evento de actualización sumando o restado respectivamente 0.5º.

Para que no se pueda interactuar desde cualquier distancia con los objetos, se ha incluido en cada uno de ellos un *Box Collider* que está activado como *trigger*. Esto permite que, cuando se entra en los límites del objeto, se habilita la posibilidad de lanzar los haces por parte del usuario.

Una vez ya se detectan y se muestran visualmente estos cambios del mundo virtual, queda crear un método PUT en la API que permita modificar los datos de la instancia del gemelo digital. Para ello, siguiendo la lógica utilizada para crear el GET en la API, se ha añadido lo siguiente:

- Añadir en el controlador un método PUT con la ruta *api/Twin/{twinId}* que llame al método del dominio *UpdateTwin*. Este método tiene como parámetro un *TwinEntity*, una entidad que permite "deserializar" el JSON de entrada en diferentes variables. En caso de que la llamada sea exitosa, simplemente se devuelve un mensaje informando de que todo ha sido correcto.

- Añadir en el dominio el método *UpdateTwin* que simplemente consulta todas las variables de *TwinEntity* y, por cada una que no tenga vacía, llama al método del setter de dicha variable de los servicios con su nuevo valor. Por ejemplo, en caso de que el JSON de la petición incluya un valor de temperatura, se llamará al método de la clase de servicios de la siguiente forma:

  *SetTemperature (twinEntity.TwinId, (double) twinEntity.Temperature)*

- Añadir en los servicios un método setter para cada una de las variables que se gestionan. Todos los métodos tienen la misma lógica, llaman al método *UpdateTwinProperty<T>,* siendo T el tipo de valor de la propiedad que actualizan. Como parámetros se deben incluir el identificador del *twin* que viene desde la petición PUT, la extensión a la propiedad dentro del *twin* y el nuevo valor. Este método se encarga de formatear el JSON según lo requiere el SDK de Azure Digital Twin e invoca al método *UpdateDigitalTwinAsync* de la clase *DigitalTwinsClient* con el identificador y valor pasados por parámetro.

En este punto, queda poder generar este PUT desde Unity. Como ya se dispone del token de autorización, obtenido al lanzar la aplicación y refrescado a través de una corrutina, solo es necesario añadir un método en el mismo script donde se llama al GET y replicarlo, únicamente añadiendo la cabecera *Content-Type* con valor *"application/JSON"* y añadiendo un parámetro que sea el *string* que contiene la información a actualizar.

Todos los objetos que cuenten con acciones de temperatura, luz u ocupación, cuando actualicen su valor después de que se produzca el haz desde el usuario y el reflejo visual de esta acción en Unity, serializaran sus valores siguiendo la estructura que corresponda en cada caso e invocaran la corrutina de *PutState*, adjuntando su nuevo estado recién actualizado.

Esto ya permite que, cada vez que se produce una interacción con un objeto, este nuevo valor se actualice en la instancia del gemelo digital. Y, tal y como se ha indicado en el contexto inicial, como los dispositivos inteligentes ya están conectados al gemelo digital por un desarrollo anterior, ya se actualizan cuando se produce un cambio, por lo que no es necesario hacer nada más.

## 6.6. Test a usuarios y clientes

### 6.6.1. Características de una muestra ideal

Para cumplir con la identificación de perspectiva del análisis de viabilidad, una muestra ideal para las pruebas representaría la diversidad en términos de edad, sexo, nivel educativo, competencias digitales y capacidades físicas.

Esta es una propuesta de muestra que encajaría en todas las categorías:

1. Género: Para representar la sociedad, se incluirían a 3-5 mujeres, 3-5 hombres y 1-2 personas de otros géneros.
2. Edad: Para captar todo tipo de edades, se podrían incluir 3-4 personas por cada franja de diez años a partir de los 20. Es decir, se harían cuatro grupos con este número de integrantes, que serían, de 20 a 30, de 30 a 40, de 40 a 50 y por encima de 50.
3. Nivel educativo: Para representar la realidad, se incluirían 5-6 con educación secundaria, 4-5 con carrera universitaria y 2-3 con máster o posteriores.
4. Discapacidad: Adjuntar a 2-3 personas con diferentes tipos de discapacidad física proporcionaría información valiosa sobre la accesibilidad y la inclusión de la herramienta.

Esta muestra diversa ayudaría a garantizar que el producto final sea utilizable y eficiente en un amplio espectro demográfico. Pero las cifras son imposibles de conseguir por disponibilidad temporal, así que las pruebas se limitarán a la muestra explicada en el estudio de viabilidad.

### 6.6.2. Descripción de test de usuario y aceptación

A continuación, se explica la definición de los test de usuario que se van a realizar las personas de la muestra seleccionada. El objetivo es evaluar la usabilidad del entorno virtual y la calidad de la réplica del espacio de oficina.

La selección de participantes es la especificada en la muestra del análisis de viabilidad: ingeniero de software de 53 años, diseñadore de experiencia de usuario de 26 años y directora de instalaciones de 38 años.

A continuación, se adjuntan las tareas a realizar por cada uno de los usuarios de la muestra:

- Tarea 1: Navegar desde la entrada virtual hasta la sala de innovación.
- Tarea 2: Interactuar con objetos digitales (abrir luz, ocupar silla, cambiar temperatura).

Mientras se realicen las sesiones de prueba, cada participante realizará las tareas en un entorno controlado mientras la encargada del proyecto observa sus acciones y registra información relevante. Después, se analizarán datos como el tiempo que han tardado en las tareas, la tasa de errores y las opiniones subjetivas de los usuarios. En base a esto, si los usuarios tienen dificultades para navegar por el espacio o interactuar con los objetos, se introducirán cambios en el diseño si da tiempo. Si no, se dejarán como posibles ampliaciones.

También se han definido cuáles serán los test de aceptación para poder desplegar el producto. Los criterios de aceptación incluyen una réplica exacta del entorno físico de la oficina, una comunicación segura en tiempo real y una interfaz de usuario intuitiva.

Para cumplir estos criterios, los casos de prueba serán:

1. Llevar a cabo varias interacciones desde el mundo virtual, probando la calidad y la latencia de la comunicación en tiempo real con el gemelo digital.
2. Llevar a cabo varias interacciones desde el mundo real, probando la latencia hasta reflejarse en el mundo virtual

El equipo de Innovación será quien ejecutará las pruebas, ya que son los clientes del proyecto. Para el caso A, una persona interactuará con el mundo virtual y otra controlará lo que sucede en la instancia del gemelo digital y cuanto tarda en llegar. Para el caso B, una persona encenderá la luz del mundo real y otra controlará cuando se refleja este cambio en Unity.

Después se registrarán tanto los procesos como los resultados, para posteriormente hacer una revisión final y, si se cumplen todos los criterios, proceder con el despliegue del producto. En caso de que no funcione, se harán las revisiones pertinentes y se volverán a realizar los test manuales.

### 6.6.3. Resultados de los test de usuario

Durante las pruebas de usuario, los participantes se enfrentaron inicialmente a dificultades para apuntar y moverse dentro del entorno virtual. Una vez que se les proporcionaron instrucciones claras, su navegación mejoró y pudieron completar con éxito las tareas asignadas. Sin embargo, un problema común fue la falta de claridad sobre con qué objetos se podía interactuar y con cuáles no. Como futuros pasos, se introducirán indicadores o señales visuales que indiquen claramente los elementos interactivos del entorno virtual.

### 6.6.4. Resultados de los test de aceptación

En las pruebas de aceptación se observó que Event Grid se saturaba cuando no se aplicaban filtros. Después de aplicarse según el identificador de los nodos relevantes, todo funcionó mucho mejor. Además, se observó que la Azure Function no funcionaba correctamente con un Service Plan gratuito. Una vez se cambió por el plan premium, la comunicación era prácticamente instantánea. Por lo tanto, después de realizar los cambios, se puede concluir que podría desplegarse la herramienta dada la superación de todos los test.

# 7. Conclusiones

Una vez terminado el desarrollo, este apartado pretende hacer una valoración general de todo el proceso, destacando los aspectos más importantes y aquellos que tienen margen de mejora. A continuación, se muestra un esquema de todos los servicios que se requieren para que este proyecto funcione:

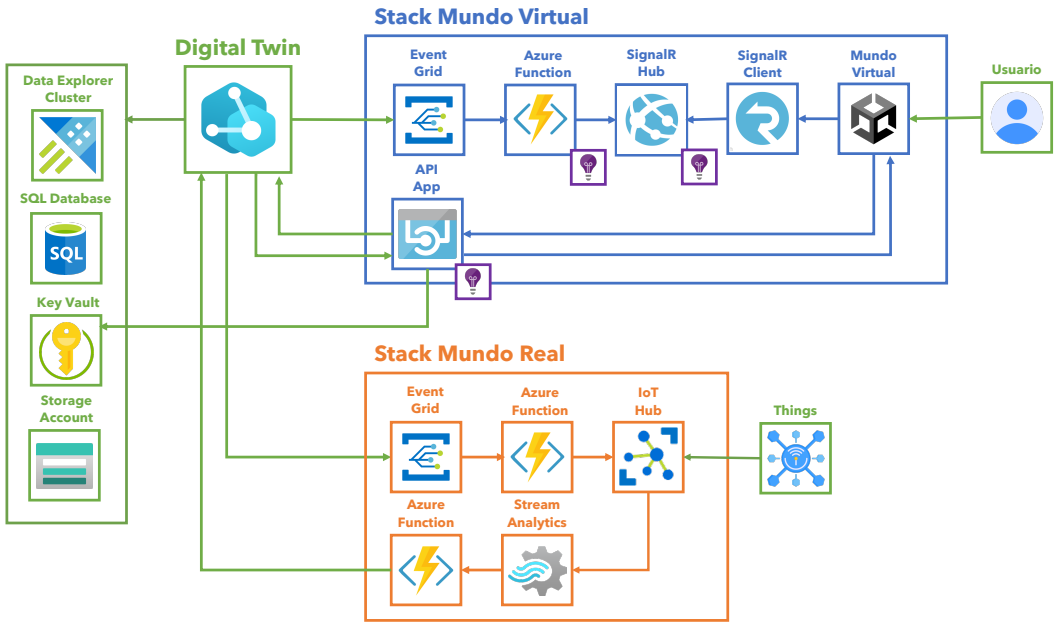


Figura 7.1. Esquema de los recursos que afectan al proyecto. Fuente: elaboración propia.

La primera conclusión interesante es que, a pesar de ofrecer una buena herramienta para la visualización de datos del gemelo digital, 3D Scene Studio se queda corto si se quieren hacer funcionalidades personalizadas. Es cierto que la herramienta está en *preview*, pero las opciones de reglas visuales y widgets son muy limitadas. Además, no hay forma de crear acciones que afecten sobre el DT, simplemente se pueden visualizar datos. Además, al añadir pocos elementos y comportamientos, la herramienta iba demasiado lenta.

Sobre la integración de los diferentes recursos de Azure, se puede concluir que ha demostrado un alto rendimiento y una integración perfecta y sencilla. La fiabilidad de Azure, evidente por la ausencia de caídas del servicio durante todo el proceso, valida la decisión de haber elegido este servicio en la nube. Sin embargo, la integración de Unity con Azure presentaba retos y complejidades que requerían conocimientos y experiencia previos en ambas plataformas, para superar así con eficacia diversos obstáculos operativos que han provocado un ralentizado en el proceso.

Haciendo hincapié en este punto, la familiaridad con el ecosistema Azure ha desempeñado un papel importante en la gestión de los servicios y la comprensión de sus funcionalidades, lo que ha permitido tomar decisiones informadas con respecto a la selección de recursos. Del mismo modo, los conocimientos previos en modelado 3D y Unity resultaron cruciales para agilizar el desarrollo del proyecto. Sin estos conocimientos, recurrir a objetos 3D que no fuesen propios habría puesto en peligro el acabado personalizado deseado y la representación exacta 1:1 del modelo del espacio de oficinas que este desarrollo ha obtenido.

Cabe mencionar que el proceso de modelado en 3D habría sido mucho más rápido con un ordenador más potente. Además, las limitaciones a la hora de instalar el paquete de herramientas de Adobe en el ordenador portátil de la empresa, ha provocado muchos inconvenientes en el flujo de trabajo, haciendo necesarias las transferencias de archivos entre los diferentes dispositivos, mayoritariamente a través de Microsoft OneDrive, debido a la cantidad y el tamaño de estos.

En cuanto al uso de SignalR como método de comunicación en tiempo real, cabe destacar que, la falta de documentación oficial para utilizarlo en Unity ha provocado que se dependiera de soluciones basadas en la comunidad y recursos alternativos, lo que ha ralentizado el progreso. La gestión manual de versiones DLL específicas para la integración en Unity resultó muy engorroso. Sin embargo, después de extensas pruebas y esfuerzos de integración, SignalR ha funcionado con éxito según lo previsto, demostrando facilidad de ampliación y un rendimiento eficaz.

El uso de Event Grid para detectar actualizaciones en el gemelo digital y las Azure Functions para la invocación de eventos remotos en SignalR, ha sido simple de programar y ha ofrecido una solución efectiva. La documentación completa y fiable de Microsoft ha desempeñado un papel crucial en la comprensión e implementación de estas herramientas con éxito.

La creación de la API ha resultado ser un proceso bien documentado y manejable, a pesar de no tener experiencia previa con C# a la hora de programar este tipo de aplicaciones. La integración con el SDK de Azure Digital Twin fue sencilla y eficaz, gracias a la extensa documentación. Sin embargo, la protección de la API y la creación de la App Registration para el acceso a servicios externos ha resultado ser más complicado y lento de lo que se podía esperar.

También, según los resultados de las pruebas de usuario, es evidente que los participantes tuvieron dificultades para identificar los objetos con los que se podía interactuar dentro del mundo virtual. Se debería haber implementado algún tipo de alerta visual pero no llegó el tiempo.

En conclusión, la evaluación de todo el proceso pone de relieve las ventajas y los retos de la integración de la comunicación entre Unity y Azure Digital Twin para la gestión y reacción bidireccional. El objetivo principal del proyecto se ha alcanzado con éxito, dando como resultado el producto propuesto. Aunque los objetivos secundarios también se han cumplido todos en cierta medida, el despliegue de la solución entre los trabajadores permitiría investigar a fondo su facilidad de uso, las mejoras de productividad y las mejoras de eficiencia energética.

# 8. Posibles ampliaciones

En esta sección, siguiendo con las conclusiones del apartado anterior, se van a hacer una serie de sugerencias en relación con cómo se podría continuar el desarrollo para mejorar el trabajo existente y que nuevas funcionalidades se podrían añadir para evolucionar el proyecto.

La primera, y más obvia, sería la ampliación de los dispositivos que se gestionan. Dado que este proyecto sirve como prueba de concepto, gestionar luces, temperatura y ocupación tiene sentido, pero en caso de seguir con el desarrollo se debería explorar la expansión tanto de dispositivos IoT como de objetos gestionados dentro del gemelo digital. Esto habilitaría una visión y control más exhaustivos de la oficina.

La segunda sería resolver la necesidad de implantar un sistema de alerta visual en los objetos con los que se puede interactuar, como un efecto de sombra, que resaltase los objetos cuando el cursor se situase sobre ellos. Esta señal visual proporcionará indicaciones claras e intuitivas a los usuarios, mejorando mucho la usabilidad de la aplicación y la experiencia de los usuarios.

La tercera seria la incorporación de datos históricos dentro del mundo virtual. Esta es una capacidad de la que dispone 3D Scene Studio y resulta muy interesante para ver la evolución de un valor en el tiempo. Aprovechando que la instancia del gemelo digital ya guarda este tipo de información en el recurso Azure Data Explorer Cluster [18], se podrían añadir en el mundo virtual para favorecer la contextualización y toma de decisiones.

La cuarta seria explorar la integración con realidad virtual o mixta (RV/RM). Para mejorar la experiencia e inmersión del usuario, se podrían añadir este tipo de tecnologías a la hora de interactuar con el mundo virtual de una forma más intuitiva y atractiva. Este paso se quería realizar en el desarrollo, pero se tuvo que descartar para priorizar otras funcionalidades primordiales.

La quinta, y última, sería la implementación de un mundo persistente, es decir, subir la instancia del mundo virtual a un servidor para mejorar la comunicación y colaboración entre trabajadores. Además, se podría incluir un sistema de comunicación a través de carteles virtuales, permitiendo así a los usuarios dejar mensajes en puntos específicos del mundo, facilitando futuras consultas y colaboraciones.

# 9. Bibliografía

[1] «Hyundai Motor se asocia con Unity para construir una metafábrica». https://www.hyundai.news/es/articles/press-releases/hyundai-motor-se-asocia-con-unity-para-construir-una-metafabrica.html (accedido 2 de febrero de 2023).

[2] M. Grieves, *Origins of the Digital Twin Concept*. 2016. doi: 10.13140/RG.2.2.26367.61609.

[3] «GE Predix Software Platform Offers 20% Potential Increase in Performance Across Customer Base; New GE Offerings Accelerate Transformation into a Digital Industrial Company | GE News». https://www.ge.com/news/press-releases/ge-predix-software-platform-offers-20-potential-increase-performance-across-customer (accedido 2 de febrero de 2023).

[4] «IBM Watson IoT Platform», 29 de septiembre de 2015. https://internetofthings.ibmcloud.com/internetofthings.ibmcloud.com (accedido 2 de febrero de 2023).

[5] «IBM Digital Twin Exchange - Overview», 15 de diciembre de 2022. https://www.ibm.com/products/digital-twin-exchange (accedido 2 de febrero de 2023).

[6] «Announcing Azure Digital Twins: Create digital replicas of spaces and infrastructure using cloud, AI and IoT | Blog y actualizaciones de Azure | Microsoft Azure». https://azure.microsoft.com/es-es/blog/announcing-azure-digital-twins-create-digital-replicas-of-spaces-and-infrastructure-using-cloud-ai-and-iot/ (accedido 2 de febrero de 2023).

[7] N. Stephenson, *Snow crash*. New York : Bantam Books, 1993. Accedido: 2 de febrero de 2023. [En línea]. Disponible en: http://archive.org/details/snowcrash00step

[8] «Meta is desperately trying to make the metaverse happen», *MIT Technology Review*. https://www.technologyreview.com/2022/10/11/1061144/metaverse-announcements-meta-connect-legs/ (accedido 4 de febrero de 2023).

[9] «IKEA VR Experience en Steam».

https://store.steampowered.com/app/447270/IKEA_VR_Experience/ (accedido 4 de febrero de 2023).

[10] «BMW Group and NVIDIA take virtual factory planning to the next level». https://www.press.bmwgroup.com/global/article/detail/T0329569EN/bmw-group-and-nvidia-take-virtual-factory-planning-to-the-next-level?language=en (accedido 2 de febrero de 2023).

[11] «Building Twin», *siemens.com Global Website*. https://new.siemens.com/global/en/products/buildings/digital-building-lifecycle/building-twin.html (accedido 2 de febrero de 2023).

[12] «AWS IoT vs. Azure IoT Hub vs. Google Cloud IoT Core vs. IBM Watson Comparison». https://sourceforge.net/software/compare/AWS-IoT-vs-Azure-IoT-vs-Google-Cloud-IoT-Core-vs-IBM-Watson/ (accedido 4 de febrero de 2023).

[13] baanders, «Ingest telemetry from IoT Hub - Azure Digital Twins», 30 de noviembre de 2022. https://learn.microsoft.com/en-us/azure/digital-twins/how-to-ingest-iot-hub-data (accedido 4 de febrero de 2023).

[14] «Home», *Scrum.org*. https://www.scrum.org/index (accedido 4 de febrero de 2023).

[15] «Azure Boards | Microsoft Azure». https://azure.microsoft.com/es-es/products/devops/boards (accedido 10 de junio de 2023).

[16] baanders, «Quickstart - Get started with 3D Scenes Studio (preview) - Azure Digital Twins», 10 de marzo de 2023. https://learn.microsoft.com/en-us/azure/digital-twins/quickstart-3d-scenes-studio (accedido 11 de junio de 2023).

[17] bradygaster, «Overview of ASP.NET Core SignalR», 14 de febrero de 2023. https://learn.microsoft.com/en-us/aspnet/core/signalr/introduction (accedido 14 de abril de 2023).

[18] baanders, «Data history (with Azure Data Explorer) - Azure Digital Twins», 9 de marzo de 2023. https://learn.microsoft.com/en-us/azure/digital-twins/concepts-data-history (accedido 12 de junio de 2023).