DESARROLLO DE UN FRAMEWORK PARA LA CREACIÓN DE GEMELOS DIGITALES USANDO IOT

CARLOS MORO GARCÍA

MÁSTER EN INTERNET DE LAS COSAS. FACULTAD DE INFORMÁTICA
UNIVERSIDAD COMPLUTENSE DE MADRID

Trabajo Fin Máster en Internet de las Cosas

Enero de 2020
Convocatoria: Enero-Febrero de 2020
Calificación: 8.5 (Notable)

Director:

Jorge Jesús Gómez Sanz

Autorización de difusión

Carlos Moro García

Enero de 2020

El abajo firmante, matriculado en el Máster en Internet de la Cosas de la Facultad de Informática, autoriza a la Universidad Complutense de Madrid (UCM) a difundir y utilizar con fines académicos, no comerciales y mencionando expresamente a su autor el presente Trabajo Fin de Máster: *Desarrollo de un framework para la creación de gemelos digitales usando IoT*, realizado durante el curso académico 2019-2020 bajo la dirección de Jorge Jesús Gómez Sanz en el Departamento de Ingeniería del Software e Inteligencia Artificial, y a la Biblioteca de la UCM a depositarlo en el Archivo Institucional E-Prints Complutense con el objeto de incrementar la difusión, uso e impacto del trabajo en Internet y garantizar su preservación y acceso a largo plazo.

Resumen en castellano

El crecimiento de las nuevas tecnologías y el Internet de las Cosas ha empujado a todos los aspectos de la sociedad a crecer a la par. En este contexto aparece la industria 4.0, que busca aprovechar los datos de sensores para mejorar procesos y descubrir problemas. Una herramienta muy útil en la industria 4.0 es el concepto de **gemelo digital**. El gemelo digital permite tener una copia exacta de cualquier objeto de manera virtual. La literatura al respecto indica que descubrir y resolver problemas en este gemelo es más eficaz.

Para estudiar el concepto, el proyecto propone una plataforma de gestión de gemelos digitales. Se ha llamado ***DgIoTwins***. En esta plataforma se diferencia entre **modelo 3D** y **escenario**. Un modelo 3D es simplemente la copia de un objeto real –por ejemplo, un aerogenerador como objeto abstracto– de forma virtual. El escenario hace referencia a un objeto específico –por ejemplo, un aerogenerador concreto de un parque eólico concreto– usando un modelo 3D y un modelo de datos con los datos de dicho objeto.

Para facilitar el acceso al gemelo digital por parte de los usuarios, se proponen también visualizadores que funcionen en un navegador web, para **realidad virtual** y **aumentada**.

La comunicación entre los objetos reales y sus respectivos escenarios se realiza mediante **MQTT** y un protocolo de aplicación propio. Este protocolo describe la semántica para sincronizar el modelo virtual y el objeto real.

Por último, se detallan las conclusiones sacadas del proyecto, los inconvenientes y las posibles líneas de trabajo futuro.

Palabras clave

Realidad virtual, realidad aumentada, 3D, IoT, MQTT, WebGL, WebVR, DgIoTwins

Abstract

The evolution of the new technologies and the Internet of Things has pushed all aspects of society, and these have evolved at the same time. In this context, industry 4.0 appears. It tries to take advantage of sensors data, to improve processes, and to find out problems. The concept of **digital twins** is a really useful tool. A digital twin is a virtual copy of an object. Literature on this subject shows that finding out and resolving problems with the twin is more efficient.

To study the concept, this project proposes a digital twins manager platform whose name is **DgIoTwins**. In this platform, we can find **models 3D** and **stages**. A model 3D is simply a virtual copy of a real object –for instance, a generic wind turbine–, and a stage is a representation of a concrete object –for instance, a particular wind turbine in a particular wind farm–. A stage consists of a model 3D and data of the particular object.

Every model 3D can be visualized in a web browser, in **virtual reality**, and in **augmented reality**.

The communication between the real objects and their respective stages is done through **MQTT** and a custom application protocol. This protocol describes the semantics to synchronize the virtual model with the real object.

Finally, this paper details the conclusion of the project, the inconveniences, and the possible future lines that the project could follow.

Keywords

Virtual reality, augmented reality, 3D, IoT, MQTT, WebGL, WebVR, DgIoTwins

Índice general

Índice de figuras

Agradecimientos

Me gustaría agradecer a mis padres por apoyarme siempre y ser la mejor ayuda a la hora de tomar decisiones. Sin ellos nunca habría acabado este proyecto.

A mis hermanos, Adriana y César, que siempre están ahí.

A todos mis amigos y amigas con los que siempre se puede desconectar un rato de las obligaciones. Especialmente a Alda, que me ha acompañado durante todo el máster siendo mi compañera en todos los laboratorios y haciéndolos mucho más divertidos.

Y a todos los profesores que me han dado clase durante este máster y de los que he aprendido todo lo que luego he podido plasmar en este trabajo.

Este trabajo se ha desarrollado en parte usando recursos del proyecto "*DISEÑO COLABORATIVO PARA LA PROMOCION DEL BIENESTAR EN CIUDADES INTELIGENTES INCLUSIVAS*" (TIN2017-88327-R)

Dedicatoria

Dedicado a todas las personas que se ven obligadas a trabajar para poder pagarse los estudios, se esfuerzan el doble y se les reconoce la mitad.

¡Por una educación pública, gratuita y de calidad!

Capítulo 1

Introducción

1.1. Motivación

En la actualidad, uno de los campos tecnológicos de desarrollo más importantes y con más proyección es, sin duda, el Internet de las Cosas (IoT). La mejora de las redes de comunicación ha facilitado su evolución y las «futuras» redes 5G van a suponer un gran impulso para esta tecnología gracias a su baja latencia y su velocidad [33].

La industria tiene un gran potencial para sacar un gran rendimiento al Internet de las Cosas, dando lugar a lo que llamamos IoT industrial (IIoT) o Industria 4.0 [6].

Entre todas las herramientas que aporta el IoT a la industria, una de las más útiles es el concepto de **Gemelos Digitales** [4, 25].

Un gemelo digital es un modelo virtual interactivo e idéntico de un proceso, servicio o producto. Sin embargo, gracias al Internet de las Cosas, un gemelo digital es mucho más que una simple copia digital; es posible sensorizar cualquier objeto, medir cada uno de sus parámetros y replicarlos de forma virtual. Y, de igual forma, interactuar con el modelo virtual y que los cambios se reflejen en el real. De esta manera se puede monitorizar el objeto para analizar su comportamiento, ya que la réplica virtual contiene toda la información del sistema físico. Según la Figura 1.1, se tiene un objeto real y un objeto digital. El objeto a visualizar puede estar ya sensorizado o no. Puede tener sensores y medios para comunicarse, o puede no tener controladores que le permitan ejecutar acciones. Cada caso plantea retos

diferentes. El gemelo digital toma esta información y la representa para un usuario. Pero también permite operar sobre el objeto 3D y modificar el real.

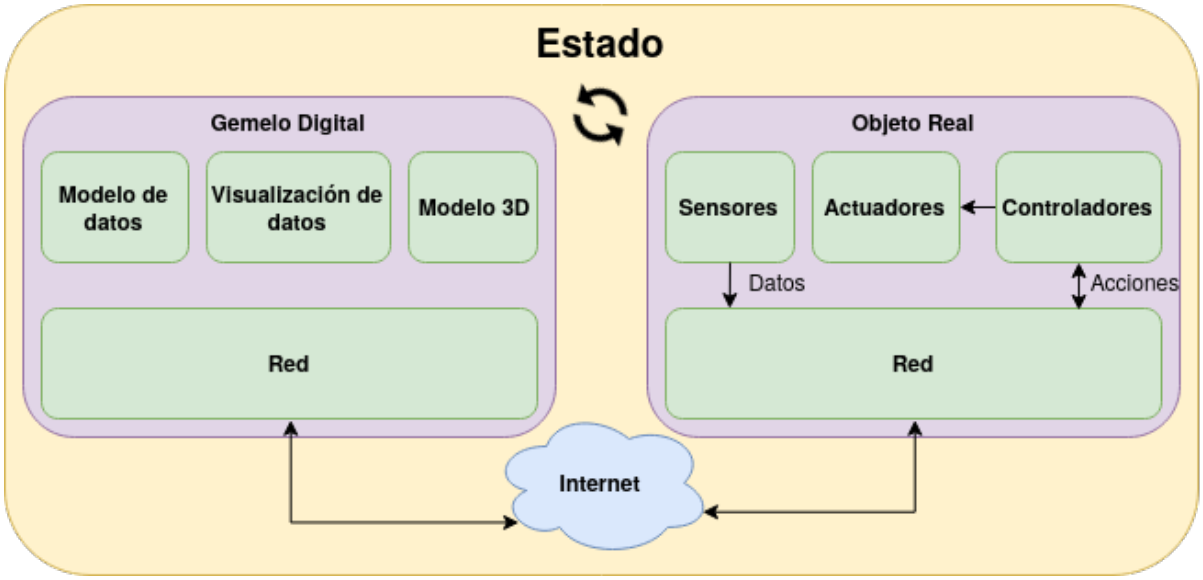


Figura 1.1: *Concepto de gemelo digital. El objeto real y su gemelo comparten el mismo estado sincronizándolo a través de Internet. El objeto real tiene varios sensores para conocer su estado, actuadores para realizar acciones y controladores, y el gemelo digital guarda los datos, los muestra y representa el objeto en tres dimensiones.*

En general, este enfoque permite el intercambio de datos y gracias a este intercambio, se puede monitorizar, controlar y automatizar actividades como encender una máquina en un momento concreto o enviar una alerta cuando la temperatura de un centro de datos (*data center*) haya superado un umbral preestablecido sin desplazarse físicamente.

Aparte, se pueden descubrir problemas antes o mejorar la eficiencia de procesos industriales o máquinas. Como resultado, se obtiene un aumento de la productividad, un fortalecimiento de la seguridad en el trabajo, una mejora de la eficiencia de la gestión de recursos, mayor control y seguimiento de la producción, y una reducción de los costes de fabricación.

Se pueden distinguir tres tipos de gemelos digitales [15, 37]: de **alta fidelidad**, implica un análisis en profundidad del objeto; **funcional**, aporta datos básicos sobre el estado del objeto, por ejemplo, apagado, encendido, lleno o vacío, y **estadístico**, se usa para recolectar datos en crudo que serán analizados posteriormente. Este proyecto se centra en

los funcionales.

Para realizar este enfoque de gemelo digital funcional, se requieren elementos que permitan captar datos en el objeto físico y transmitirlos al gemelo digital. Estos elementos se enumeran a continuación:

- **Dispositivos inteligentes**. Pueden ser sensores, actuadores o controladores. Deben ser de bajo coste y consumir poca energía. Son los encargados de recoger lo datos, enviarlos y ejecutar las acciones pertinentes. Son necesarios para convertir un objeto normal en uno inteligente en los casos en los que se requiera. Como se ve en la Figura 1.1, son una parte fundamental del objeto real.

- **Comunicación**. Son imprescindibles redes, protocolos e interfaces de comunicación para conectarse a Internet y compartir los datos de los dispositivos entre ellos y con el servidor central que aloja al gemelo. Sin comunicación no se podrían sincronizar el objeto real con su gemelo.

- **Infraestructura de datos**. Se necesita una infraestructura que concentre los datos de todos los sensores y los guarde. Sin una infraestructura que almacene los datos sería imposible guardar y sincronizar el estado del objeto y su gemelo. Suele ser un servicio en la nube, ya sea un servidor propio o una nube pública.

- **Interfaz de usuario**. Aparte de la comunicación entre máquinas, es importante que un usuario humano puede ver los datos e interactuar con ellos. La mejor manera de representar tan alta cantidad de datos es mediante gráficas, alertas y parámetros estadísticos. En este apartado es donde se encuentra la representación 3D de los gemelos digitales. Para visualizar los gemelos digitales existen varios métodos distintos según necesidades y tecnología [38]: centro de visualización, realidad virtual, realidad aumentada y realidad mixta.

- Como un extra se podría encontrar un **analizador de datos**. La mayoría de los datos no tienen valor *per se*, se necesita una aplicación que analice los datos en crudo y los

convierta en información relevante. Este elemento le añade valor al gemelo digital, pero no es imprescindible para desarrollar su cometido. Sería muy interesante mezclar los gemelos digitales con técnicas de *big data* y *machine learning* para procesar los datos en tiempo real y que el modelo virtual sea capaz de tomar sus propias decisiones.

Combinando estos elementos, se debería poder sincronizar el objeto real y el modelo virtual para que ambos realicen las mismas acciones. Pero sin olvidar que hay problemas que hay que solucionar en primer lugar, como el renderizado de modelos 3D de manera virtual, la interacción con ellos y la comunicación de la plataforma con el objeto real.

1.2. Objetivos del proyecto

El principal objetivo de este proyecto es desarrollar un *framework* para crear y visualizar gemelos digitales mediante el uso de IoT. Los modelos 3D se visualizarán en el navegador gracias a three.js [19], en realidad virtual a través de WebVR[1] y en realidad aumentada con AR.js [7].

Entre los principales objetivos del proyecto se pueden destacar:

- **Visualizar** modelos virtuales en 3D de objetos reales. Que además se muestren de varias formas distintas: en el navegador, a través de **realidad virtual** y a través de **realidad aumentada**. Y ser capaz de **interactuar** con ellos.
- **Sincronizar** el modelo con el objeto para que ambos actúen de la misma manera. Por lo que deben ser capaces de **comunicarse** entre ellos.
- **Gestionar** el efecto de las acciones sobre el gemelo. La plataforma permitirá definir qué animaciones se dispararán en función de las acciones definidas.

Para probar que funciona, se plantea una prueba de concepto donde se muestrean sensores simples y se actúa sobre actuadores como motores o leds. El proyecto no propone formas

[1] https://webvr.info/

concretas en que se pueden programar los sensores o actuadores de los elementos electrónicos asociados al objeto real, pero se aportan ejemplos de cómo lograrlo y qué elementos arquitectónicos serían necesarios.

El código de la plataforma y de los casos de estudio se pueden encontrar en GitHub, de la plataforma en la rama «dev» de «https://github.com/carlosmg95/TFM_Digital_Twins/tree/dev» con licencia MIT y los casos de estudio en «https://github.com/carlosmg95/TFM_Digital_Twins_Casos» sin licencia.

1.3. Estructura de la memoria

Esta sección está dedicada a dar una visión de alto nivel de cada uno de los apartados de este documento. La estructura es:

El ***Capítulo 1*** es la introducción del proyecto y describe la motivación, los objetivo principales y la estructura de la memoria.

El ***Capítulo 2*** ilustra el estado del arte de las tecnologías usadas en el proyecto. Se analizarán estas tecnologías con el objetivo de formar una base sobe ellas y dar un contexto a la idea principal.

El ***Capítulo 3*** explica la semántica más importante de este proyecto. Principalmente, lo que es un objeto real, un modelo virtual y la comunicación entre ambos para sincronizarse.

El ***Capítulo 4*** muestra los requisitos necesarios para desarrollar el proyecto y las soluciones elegidas.

El ***Capítulo 5*** muestra la arquitectura detallada de la plataforma desarrollada para este Trabajo de Fin de Máster.

El ***Capítulo 6*** muestra la interfaz de usuarios de la plataforma desarrollada.

El ***Capítulo 7*** desarrolla dos casos de uso desde su configuración hasta su ejecución para que se comprenda mejor el concepto global.

El ***Capítulo 8*** resume las conclusiones a las que se han llegado, los problemas encontrados y el trabajo futuro.

El ***Capítulo 9*** es el Capítulo 1 en inglés.

El ***Capítulo 10*** es el Capítulo 8 en inglés.

Capítulo 2

Estado del arte

2.1. Introducción

En este capítulo se destacan varios *frameworks* existentes sobre gemelos digitales, se compararán con el *framework* de este proyecto y se comentarán algunas carencias que se intentarán satisfacer. Más adelante, se explicarán las principales tecnologías y recursos que se han utilizado durante el desarrollo de este proyecto y que han permitido que se pueda realizar. En este análisis se exploran qué requisitos son necesarios revisando plataformas existentes, qué dispositivos se han usado y qué tecnologías hay que ayuden a visualizar elementos 3D de forma estética y con animaciones.

En todos los *frameworks* existentes que se han estudiado en este capítulo se pueden observar ciertas similitudes, en algunas ocasiones solo se diferencian en algunos detalles. El *framework* propuesto en este proyecto también comparte muchas de esas características y, además, permite al usuario visualizar su propio objeto en el navegador, en realidad virtual y en realidad aumentada; sincroniza el objeto real con el virtual para que actúen de la misma manera y permite interactuar con el modelo virtual.

2.2. Frameworks

En esta sección se explicarán tres *frameworks* de gemelos digitales: *Oracle IoT Digital Twin*, *Eclipse Ditto* y *Azure Digital Twins*. De estos tres *frameworks* se puede resumir que

todos se centran en los datos y en usarlos para representar un entorno físico real. Alguno de ellos permite predecir datos futuros, ya sea de manera independiente o conectando con otros servicios del mismo proveedor. También permiten observar en tiempo real qué está pasando en el objeto real. Sin embargo, ninguno de los estudiados permite visualizar el propio objeto que se está monitorizando, la representación virtual queda reducida a un conjunto de datos.

2.2.1. Oracle IoT Digital Twin

Oracle IoT Digital Twin [5] es un *software* de propietario (es decir, de código cerrado) de Oracle Corporation y necesita conectarse a la nube de la misma compañía: *Oracle IoT Cloud Service*. Su implementación se cimienta en tres pilares, cada uno de ellos se utiliza en en casos diferentes y tienen sus propias ventajas.

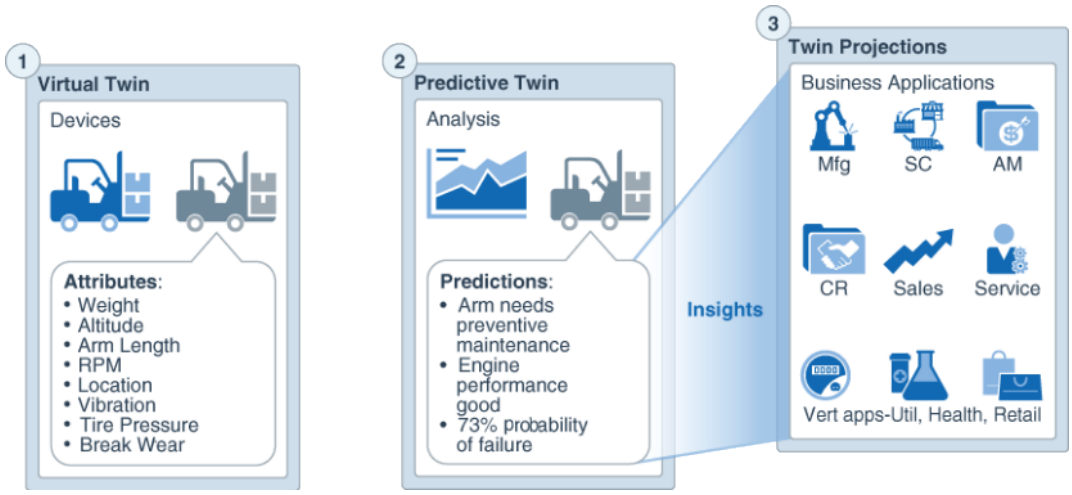


Figura 2.1: *Tres pilares de Oracle IoT Digital Twin. En el primer pilar se ve el gemelo virtual, en el segundo, el gemelo predictivo, y en el tercero, las proyecciones del gemelo [5].*

El primer pilar hace referencia al **gemelo virtual**. Es la representación virtual de un dispositivo o activo físico en la nube. Se basa en un modelo JSON con los atributos observados y deseados. Es su modelo semántico. Se usa con el objetivo de abstraerse del objeto real, ya que estos suelen tener diferentes protocolos y métodos para conectarse a la red IoT. Este módulo semántico permite especificar el rango de operación normal.

En el segundo pilar aparece el **gemelo predictivo**. Este tipo de gemelo crea un modelo

analítico y estadístico de predicción usando técnicas de *machine learning* sobre lo que le podría pasar al objeto real. A diferencia del gemelo virtual, este no es estático ni complejo y se puede adaptar a un entorno en constante cambio.

Se usa para detectar problemas futuros o el próximo estado de una máquina, de este modo es posible prepararse para cualquier situación antes de que ocurra. Además, permite identificar tendencias y patrones que pueden ser de mucha ayuda para entender el funcionamiento y el comportamiento de las máquinas.

El tercer y último pilar trata sobre las **proyecciones del gemelo**. Se usa para generar *insights* en la plataforma IoT e integrarlos con los procesos del negocio, permitiendo ejecutar acciones para arreglar el flujo de trabajo. Su principal ventaja es permitir a las aplicaciones *backend* del negocio integrarse con el sistema IoT para transformarse en sistemas inteligentes.

Lo que en el *framework* de Oracle llaman «gemelo virtual» es lo que en el *framework* de este proyecto (*DgIoTwins*) se llama «modelo de datos». *DgIoTwins* no es capaz –en esta primera versión– de predecir datos. Sin embargo, aporta la visualización del objeto en 3D, la modificación de algunos parámetros del objeto, la capacidad de ver en tiempo real cómo se está comportando mediante animaciones, que también se pueden modificar, y de controlarlo. Otra de las diferencias principales es que *DgIoTwins* es un *framework standalone* que permite conectarse con cualquier dispositivo IoT mientras que *Oracle IoT Digital Twin* está pensado para integrarse con otros servicios de la nube de Oracle.

2.2.2. Eclipse Ditto

Eclipse Ditto [8] es un *software* de código abierto de la empresa *Eclipse Foundation*. Se puede descargar e instalar tanto con Apache Maven como con contenedores Docker, y la comunicación se realiza mediante Kafka, AMQP, MQTT, HTTP o WebSockets a través de una API REST, por lo que cualquier dispositivo se puede conectar.

Este *framework* considera que los gemelos digitales son un concepto para abstraer el objeto real, pero con todas sus capacidades. El gemelo digital copia al objeto real, provee

servicios alrededor del objeto, mantiene al gemelo y al objeto sincronizados y se puede aprovechar tanto en la industria como a nivel consumidor. Según *Eclipse Ditto*, un *framework* de gemelos digitales debe ser capaz de interactuar con el gemelo digital, asegurarse de que solo usuarios autorizados pueden acceder al gemelo, poder interactuar con varios gemelos digitales e integrarse en una infraestructura *backend*.

Eclipse Ditto crea un gemelo digital como un archivo JSON con «atributos» para metadatos estáticos y «características» para datos de estado dinámicos. Los atributos pueden tener tantas claves JSON como sea necesario y tantos valores como se necesiten. Con las características ocurre lo mismo, pero con la condición de que cada característica tenga un objeto JSON llamado «*properties*». A cada uno de los objetos se les consideran una «Cosa» (*Thing*). Cada «Cosa» se compone de un id, una lista de control de acceso, los atributos y las características.

La manera de generar un gemelo digital es totalmente diferente en *Eclipse Ditto* y en *DgIoTwins*, pero con algunas similitudes en la forma de crear el fichero JSON que representa uno. Igual que en la comparación con *Oracle IoT Digital Twin*, *DgIoTwins* aporta la visualización del objeto en 3D, la modificación de sus parámetros, la capacidad de ver en tiempo real cómo se está comportando mediante animaciones y de controlarlo. En este caso, ambos *frameworks* son *standalone* que se pueden conectar con cualquier dispositivo IoT.

2.2.3. Azure Digital Twins

Azure Digital Twins [2] es un servicio en versión preeliminar de Azure IoT para crear modelos de un entorno físico. Es un *software* de propietario que se conecta con otros servicios de Azure IoT. Permite consultar datos como un conjunto como si fuesen muchos sensores independientes. Lo que permite crear una experiencia reutilizable, escalable y consciente del espacio de transmisión de datos entre el mundo físico y digital. *Azure Digital Twins* puede resultar muy útil para predecir las necesidades de mantenimiento de una fábrica, para analizar los requisitos de energía en tiempo real, optimizar el uso del espacio, realizar

un seguimiento de distintos parámetros, supervisar rutas de drones, identificar vehículos autónomos, analizar niveles de ocupación de un edifico, encontrar la máquina registradora más ocupada, etc.

Sus principales capacidades son:

- El **grafo de inteligencia espacial** es una representación virtual de un entorno real. Se utiliza para modelar las relaciones entre ambos entornos.
- Los **modelos de objetos de gemelos digitales** son protocolos de dispositivos y esquemas de datos predefinidos. Acelera y simplifica el desarrollo mediante la alineación de las necesidades específicas del dominio de la solución.
- Se pueden crear soluciones para escalar de forma segura varios **inquilinos múltiples y anidados**. Y a su vez, también se pueden crear subinquilinos de forma aislada y segura. Un inquilino es una instancia de una organización dentro de un servicio de Azure Active Directory y que se crea al registrar un servicio en la nube de Microsoft. Cada inquilino es distinto e independiente al resto de inquilinos.
- Se usan **funcionalidades de proceso avanzadas** para mejorar la personalización y automatización de las tareas de un dispositivo. Se definen y ejecutan funciones personalizadas con los datos de un dispositivo de entrada y se envían señales a puntos de conexión predefinidos.
- Tienen un **control de acceso integrado** por el cual se administra la identidad de los usuarios mediante roles y los dispositivos para controlar de forma segura su acceso.
- Por último, se pueden conectar a muchos servicios del **ecosistema** de *Azure* y *Microsoft* como *Office 365*, *Azure Maps*, *Azure AI*, *Azure Stream Analytics*...

Este *framework* ayuda a mostrar el mundo físico y sus relaciones. Simplifica el modelado, el procesamiento de datos, el control de eventos y el seguimiento de dispositivos.

2.3. Tecnologías y recursos

Para poder mostrar el objeto de modo virtual ha sido necesario revisar y adaptar las técnicas y tecnologías que se explican a continuación. También se explicarán los dispositivos usados en los casos de estudio.

2.3.1. 6LoWPAN Clicker

6LoWPAN Clicker [23] es una placa compacta de desarrollo creada por MikroElektronika[1]. Cuenta con un mikroBUS™que le proporciona conectividad física y un microchip PIC32MX470F512H [22]. Para conectarse de forma inalámbrica con otros dispositivos usa la tecnología 6LoWPAN.

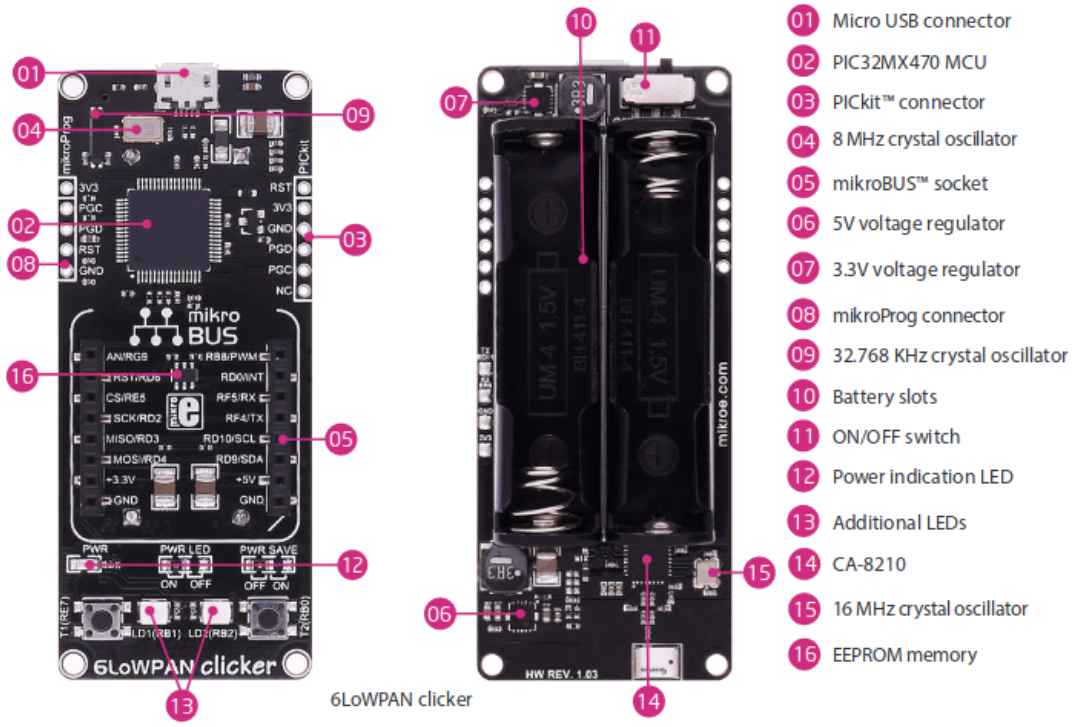


Figura 2.2: *6LoWPAN Clicker con una leyenda de cada una de sus partes. Imagen de [23].*

Se puede expandir gracias a que se le pueden conectar *click boards*™. Hay cerca de 300 diferentes entre los que se puede encontrar un sensor de temperatura, un sensor de

[1] https://www.mikroe.com/

movimiento, un acelerómetro o una matriz de leds entre otros. También se le pueden conectar otros dispositivos a través de los buses SPI e I^2C.

La placa usa un sistema operativo en tiempo real (RTOS) de código abierto llamado Contiki OS[2]. Para el control del mikroBUS™y de los *click boards*™usa la biblioteca LetMeCreate[3]. Esta biblioteca también se usará en la placa Creator Ci40.

2.3.2. Creator Ci40

La placa Creator Ci40 [20] está pensada para el desarrollo de proyectos IoT de bajo consumo. Sus interfaces de red inalámbrica son WiFi, Bluetooth y 6LoWPAN; lo que la hace perfecta para trabajar como *gateway* junto a los 6LoWPAN Clickers. Además tiene conexión Ethernet.

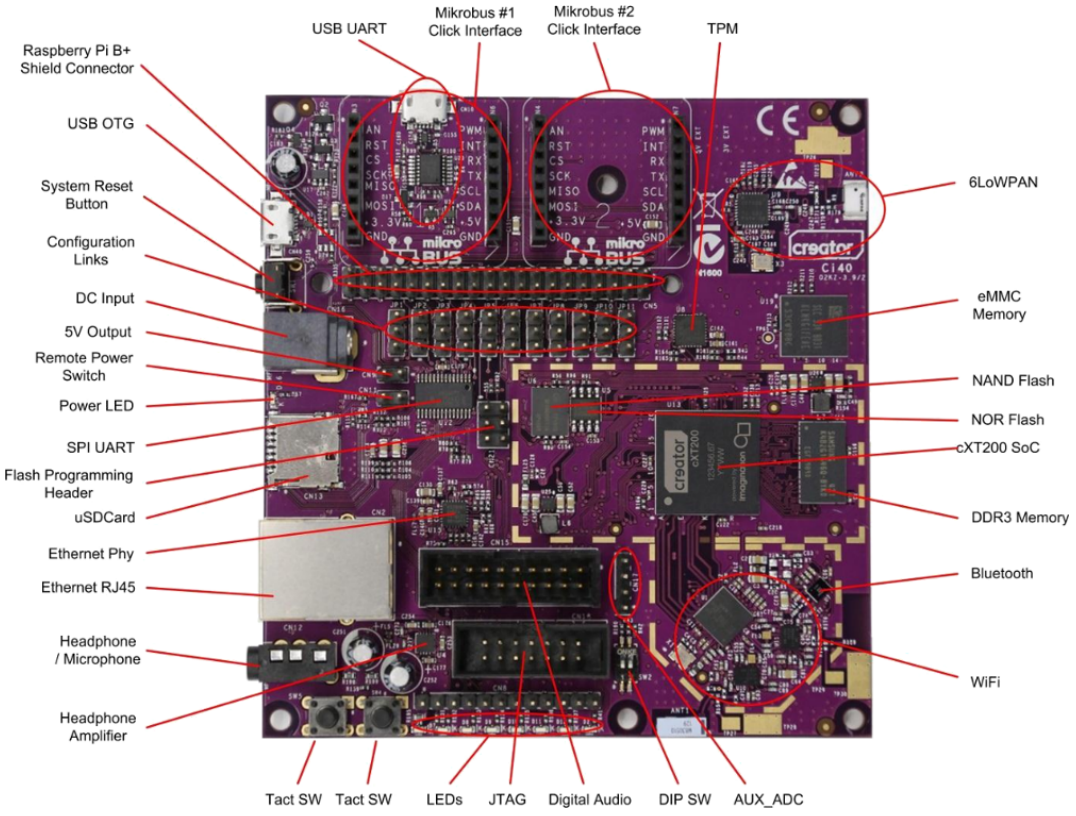


Figura 2.3: *Placa Ci40 con una leyenda de cada una de sus partes. Imagen de [20].*

[2] http://www.contiki-os.org/
[3] https://github.com/CreatorDev/LetMeCreate

Esta placa también se puede extender con los mismos *click boards*™que el 6LoWPAN Clicker.

2.3.3. Modelado y visualización 3D

El modelado 3D es una técnica de desarrollo para representar de forma tridimensional cualquier objeto y cuyo resultado es un modelo 3D.

La forma esencial de un modelo 3D es una **malla** –o *mesh*– que está formada, como mínimo, por una **geometría** y un **material**. Las figuras más básicas de geometrías son cubos, esferas, cilindros, planos, etc. Con el material se puede hacer referencia al color o, de un modo algo más complejo, añadir texturas.

Para poder mostrar el modelo 3D se necesitan tres conceptos básicos: una escena, una cámara y un *renderer*. La **escena** está formada por una o varias mallas, se le puede añadir un fondo y es necesario añadir **iluminación**, si no, no se vería el material del objeto. La **cámara** va a permitir poder observar la escena desde un punto de vista concreto, una distancia preestablecida y el ratio de aspecto elegido. Con la cámara y la escena listas, se lanza un ***renderer***, que es el encargado de generar la secuencia de imágenes vistas desde la cámara de las mallas de la escena bajo la iluminación definida.

Para el proyecto, se asume que cámara, escena e iluminación vienen ya definidas. Entonces, interesan elementos capaces de hacer el *render* de estos elementos de forma portable, preferiblemente desde el navegador. A continuación, se listan los revisados para este proyecto.

2.3.3.1. Blender

Blender [3] es un programa gratuito y de código abierto para el diseño de objetos en tres dimensiones. Permite realizar cualquier tarea relacionada con el diseño 3D: modelado, *rigging*, animación, simulación, renderizado, composición y seguimiento de movimiento, incluso edición de vídeo y creación de videojuegos.

Blender cuenta con dos motores de renderización [26], uno nativo llamado «Blender

Render» que fue creado para ser veloz, y otro llamado «*Cycles*», que es de mucha más calidad pero necesita más recursos *hardware*. Mientras que el motor interno únicamente simula la iluminación directa y las sombras, *Cycles* calcula el camino de la luz de la cámara a la fuente de luz, calculando rebotes, refracciones y alteraciones de color.

2.3.3.2. WebGL

Para mostrar los modelos 3D se usa WebGL (*Web Graphics Library*) [13]. WebGL permite renderizar gráficos 3D en cualquier navegador web mediante una API desarrollada en JavaScript.

Esta biblioteca permite la creación de modelos 3D a partir de la definición de sus vértices. Esta manera puede ser útil para crear formas muy básicas como cubos, pero para formas más complicadas, como puede ser un molino de viento, es necesario importarlas usando bibliotecas como three.js o A-Frame y crearlas con programas como Blender.

2.3.3.3. Three.js

Three.js [19] es una biblioteca que se usa junto con WebGL para mostrar gráficos animados en el navegador web en 3D. Con three.js se pueden cargar modelos 3D de varios formatos.

A parte de crear y visualizar escenas 3D, también permite gestionar y mostrar animaciones, que serán una parte muy importante del proyecto. No todos los formatos de modelos 3D permiten animaciones, es por eso que en el proyecto solo se podrán usar modelos en formato: BVH, Collada, FBX, GLTF, MMD y SEA3D.

2.3.3.4. A-Frame

A-Frame [21] es otro *framework* que permite trabajar con modelos 3D de una forma mucho más sencilla mediante el uso de etiquetas HTML. Por debajo usa three.js y permite abstraerse de las labores más complicadas. Sin embargo, esta abstracción también provoca que estemos más limitados a la hora de realizar varias tareas, por ejemplo, no se pueden

combinar varias animaciones en una sola para formar una personalizada; es por eso que para el proyecto se ha elegido utilizar three.js.

2.3.3.5. GLTF

GLTF (*GL Transmission Format*) [17]. Este formato está desarrollado por Khronos Group, el mismo grupo que desarrolla WebGL. Sus propios creadores lo definen como «el JPEG de 3D» y está pensado para ser muy eficiente, comprimiendo las escenas y modelos 3D y minimizando el procesamiento en tiempo de ejecución. El tamaño de un fichero que contiene una malla puede ser muy grande. Como se quiere trabajar en el navegador, reducir el tamaño del fichero es relevante. Por ello se estudia GLTF.

Los ficheros GLTF tienen tres posibles extensiones:

- **.gltf separado** Tiene formato JSON y necesita aparte un fichero .bin con datos importantes y otro con las texturas. Los archivos con esta clase MIME no son aceptados ya que la plataforma solo permite subir un fichero por modelo.
- **.gltf embebido** También tienen formato JSON pero se genera un solo fichero con todos los datos, por lo que sí son aceptados.
- **.glb** Tienen un formato binario y llevan toda la información, por lo que también son aceptados.

La última versión estable de Blender (2.79b) –en el momento de escribir esta memoria– no permite directamente la exportación de modelos en formato GLTF, pero se le puede añadir un *addon* [12] que lo permite.

2.3.4. Realidad virtual

La realidad virtual [35] (VR) es, según la Real Academia Española, una «representación de escenas o imágenes de objetos producida por un sistema informático, que da la sensación de su existencia real». En este proyecto, la realidad virtual es otro modo de visualizar los

modelos 3D que permite al usuario inmergirse en un entorno generado de forma artificial. Para introducirse en la realidad virtual es necesario, como mínimo, unas gafas de realidad virtual, que pueden ser acompañadas de otros accesorios como cascos, guantes o mandos.

2.3.4.1. Tipos de gafas

Todos los tipos de gafas de realidad virtual [18] se pueden agrupar en función del equipo en el que se vayan a usar: ordenador, consola o *smartphone*. Recientemente han empezado a comercializarse gafas *standalone* que no necesitan de ningún dispositivo «madre» para usarse.

El ordenador es la plataforma que más ha apostado por la realidad virtual, que está principalmente orientada al mundo del videojuego. Las gafas más destacadas [18] son las Oculus Rift [28] y las HTC Vive [16].

Para consolas la más destacada [18] es Playstation VR [34] para Playstation 4. Tanto las gafas para consola como para ordenador se deben conectar, mediante cable o manera inalámbrica, al ordenador o consola que envía la imagen y la muestra por una pantalla que está en las propias gafas.

Las gafas para *smartphone* se pueden dividir a su vez en dos categorías: gafas como las Gear VR [32] de Samsung que incluyen sensores y controladores propios, pero se limitan a los *smartphones* de la compañía, y gafas que son únicamente una carcasa con dos lentes donde se puede introducir cualquier teléfono móvil. La segunda opción es la más económica donde podemos encontrar incluso gafas de cartón como las Google CardBoard [11]. Para este proyecto se ha usado unas gafas de este segundo tipo y un *gamepad* Bluetooth de la marca Pasonomi como los que se ven en las Figuras 2.4 y 2.5.

2.3.4.2. Grados de libertad

En un movimiento tridimensional existen seis grados de libertad –*degrees of freedom* o DoF–: la traslación en cada uno de los tres ejes y la rotación en cada uno de los tres ejes. En la Figura 2.6 se detallan estos movimientos.

Figura 2.4: *Gafas de realidad virtual.*

Figura 2.5: *Gamepad para realidad virtual.*

Las gafas de realidad virtual se pueden dividir en dos en función de los grados de libertad que permiten: 3DoF, que permiten los tres movimientos relativos a la rotación, y 6DoF, que permiten todos los grados de libertad. Es decir, la sensación con gafas 3DoF es como si se estuviese quieto y solo se pudiese mover el cuello y la sensación con gafas 6DoF es mucho más inmersiva, ya que permite moverse por la escena. Las gafas con las que se ha probado este proyecto son 3DoF.

2.3.4.3. WebVR

WebVR [39, 40] es una API experimental Javascript, la cual proporciona soporte para dispositivos de realidad virtual, como el HTC Vive, Oculus Rift o Google Cardboard, en un navegador de web. Para usarlo solo se necesitan unas gafas de realidad virtual, un mando que lo controle (opcional) y un navegador compatible.

La API se está diseñando con los siguientes objetivos:

- Detectar los dispositivos de Realidad Virtual disponibles.
- Obtener las características de los dispositivos.
- Obtener la posición y orientación del dispositivo.
- Mostrar imágenes en el dispositivo con la frecuencia de refresco adecuada.

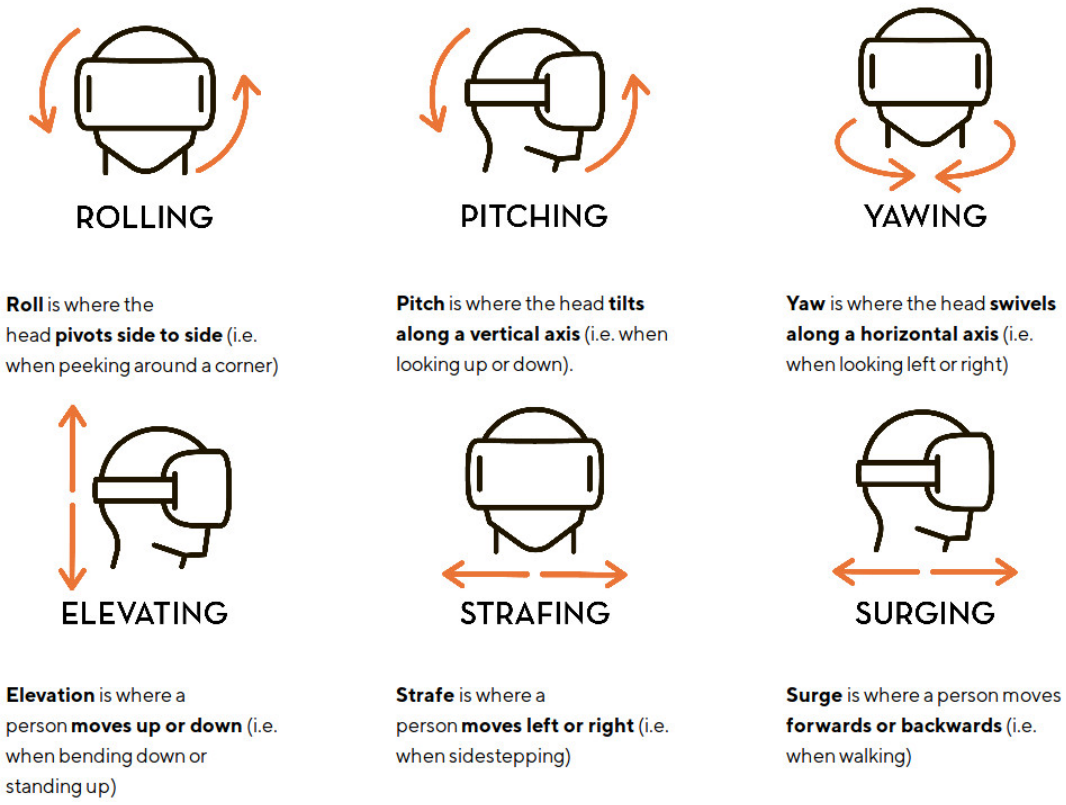


Figura 2.6: *Grados de libertad. Cada grado de libertad está acompañado de una imagen y una breve descripción [36].*

La API WebVR expone unas cuantas interfaces que permiten a las aplicaciones web incluir contenido en realidad virtual. Para conseguirlo se usa WebGL.

Los pasos necesarios para el funcionamiento de la API son:

1. Obtener una lista de los dispositivos disponibles.

2. Comprobar si el dispositivo deseado soporta los modos de presentación que requiere la aplicación.

3. Notificar al usuario la posibilidad de usar funcionalidad en realidad virtual.

4. El usuario realiza una acción para indicar que desea entrar en el modo de realidad virtual.

5. Obtener una sesión de WebVR y presentar el contenido.

6. Comenzar un bucle de renderizado que produzca los fotogramas que deben mostrarse en el dispositivo.

7. Seguir produciendo fotogramas hasta que el usuario indique que desea salir del modo de realidad virtual.

8. Finalizar la sesión de WebVR.

La parte de VR se probado con Google Chrome para Android versión 74+. La escena en realidad virtual también se muestra a través de three.js.

2.3.5. Realidad aumentada

La realidad aumentada [1] (AR) es una tecnología por la cual es posible colocar modelos virtuales sobre la realidad a través de una pantalla. Es decir, se necesita una cámara que grabe el mundo real y a través de una pantalla vemos lo que graba la cámara más el modelo virtual, a diferencia de un holograma que se ve directamente sin necesidad de pantallas. La gran diferencia con la realidad virtual es que en la realidad virtual es el usuario el que se introduce en un mundo virtual y en la realidad aumentada es el mundo virtual el que se introduce en la realidad del usuario.

2.3.5.1. AR.js

AR.js [7] es una biblioteca desarrollada para facilitar el uso de la realidad aumentada en la web. Las escenas, de nuevo, se muestran gracias a three.js y es necesario un marcador en el cual aparecerá la escena.

El patrón de la biblioteca se puede cambiar por uno personalizado, en esta plataforma se ha usado el que aparece en el Apéndice A. Lo más importante a la hora de usar un patrón es dejar un borde blanco alrededor, si se imprime y se recorta por la parte negra no va a funcionar.

Capítulo 3

Semántica del Gemelo Digital

3.1. Introducción

En el proyecto, un problema nuclear es sincronizar las acciones y eventos de un objeto real con su modelo virtual y poder visualizar en el modelo virtual los cambios que se dan en el objeto real. La semántica asociada a estos cambios se explica en este capítulo. Para ello, se define terminología y se define también la transición de estados esperada en el objeto real y en el modelo virtual.

Se entiende un **evento** como una interacción sobre el objeto, por ejemplo, pulsar un botón o la finalización de un temporizador. Si el evento se ejecuta directamente sobre el objeto se considera un **evento *hardware*** y si es sobre el modelo, un **evento virtual**. Y una **acción** es cualquier actividad o movimiento del objeto, por ejemplo, encender una luz o rotar un eje. Al igual que ocurre con los eventos, una acción ejecutada en el mismo objeto se considera una **acción *hardware*** y cuando se ejecuta en el modelo se considera una **acción virtual**. Todos los eventos y acciones *hardware* tienen su réplica virtual. Además, si el objeto es capaz de tomar **medidas** de algún tipo –como la temperatura o el flujo de corriente eléctrica– tendrá que enviársela al modelo virtual y que este la muestre. En el modelo virtual, el efecto de las acciones sobre el objeto real se representan con gráficas construidas sobre estas medidas y con animaciones preprogramadas.

3.2. Mecánica de la sincronización

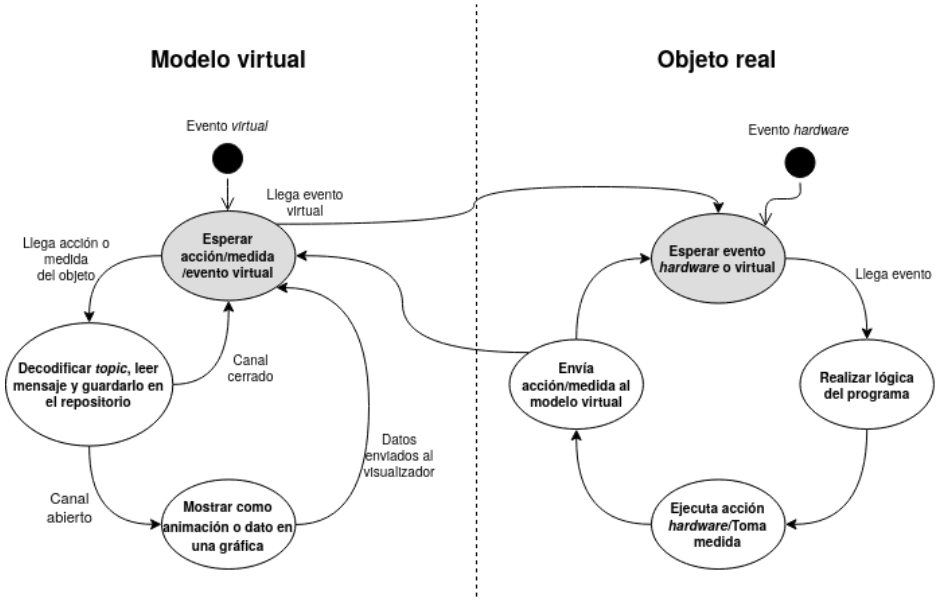


Figura 3.1: *Flujo de sincronización entre un objeto real y su modelo virtual*

La Figura 3.1 muestra, de una forma simplificada, cómo se realiza la sincronización. El **objeto real** es el que en todo momento decide cómo actuar, en él está la lógica del programa. Cuando recibe un evento, ya sea directamente (*hardware*) o desde su réplica del modelo (virtual), ejecuta un código programado que le hace realizar una acción (*hardware*) y además se la debe comunicar al modelo para que realice la misma acción (virtual).

En el otro lado, es necesario vincular un **modelo virtual** que guarde los datos e imite al objeto. El modelo está a la espera de eventos virtuales que se dan cuando un usuario interactúa con él. Una vez le llega se lo comunica al objeto real para que este decida cómo actuar. También espera las posibles acciones o medidas que le manda el objeto real. Cuando le llegan, decodifica el mensajes y guarda los datos. En el caso de que un usuario tenga abierto el canal de visualización, muestra la acción mediante una animación en el mismo momento o añade la medida a su gráfica. Para que el modelo funcione es necesario que anteriormente se programe correctamente para escuchar los eventos, acciones y medidas adecuados.

3.2.1. Modelos virtuales

Para poder hacer uso de los eventos, cada modelo 3D subido a la plataforma debe estar dividido en varias mallas, por lo menos, que las zonas donde se quiere recrear un evento sea una malla independiente al resto. Por ejemplo, hay un mando con un botón y se quiere recrear el evento de pulsar el botón desde el modelo, para que la plataforma sepa que se está pulsando en el botón y no en otra parte del mando el modelo debería estar dividido en la malla «base» y la malla «botón». Para usar las acciones, cada modelo 3D subido tiene que tener configuradas ciertas **animaciones simples** que luego en la plataforma se podrán combinar y configurar para formar acciones. Por ejemplo, el modelo 3D de un coche tiene configuradas las animaciones correspondientes a girar cada rueda y en la plataforma se pueden combinar y configurar para que giren indefinidamente y así se recrea la acción «conducir».

El siguiente paso se realiza en la plataforma, y consiste en crear dentro del escenario los mismos eventos que puede experimentar el objeto, por ejemplo, pulsar un botón. También hay que crear las acciones que se ejecutan en el objeto, por ejemplo, conducir. Es importante crearlo porque el modelo solo va a ser capaz de enviar los mensajes correspondientes a los eventos creados y solo va a escuchar a la espera de las acciones creadas.

3.2.2. Objetos reales

La lógica del funcionamiento está embebida en el objeto real. En este proyecto, hay que programar como se va a responder a eventos, qué eventos tienen prioridad (virtuales o físicos), cuándo se va a realizar una acción, etc.

El modelo virtual de la plataforma no tiene nada programado, únicamente comunica eventos cuando se interacciona con él y está a la espera de recibir datos. Si un programa dice que si se pulsa un botón se enciende un led y se pulsa el botón del modelo virtual, el modelo virtual lanzará el evento «botón pulsado», el objeto real decide qué hacer –en este caso encender un led– y una vez originada la acción en el objeto real envía un mensaje a la

plataforma para que realice la misma acción.

Para poder interactuar con la plataforma y ser monitorizados de esta forma, los objetos reales deben cumplir con una serie de características y tienen que enviar los datos de una manera determinada. Además de estar escuchando a la plataforma para un posible evento.

Se pueden distinguir cuatro tipos de objetos reales diferentes, aunque para la plataforma son totalmente transparentes.

1. Primero, se pueden encontrar objetos reales que son de por sí «completos», es decir, son capaces de recolectar datos por sí mismos y comunicarse. Un ejemplo de este tipo de dispositivos es un termostato inteligente.

2. El segundo tipo de objetos pueden tomar medidas pero no las comunican, por ejemplo, la depuradora de una piscina midiendo el pH. Estos objetos necesitan que se les añada un módulo aparte con conexión a Internet.

3. El tercer tipo son objetos cuya actividad se quiere medir o controlar pero que no toman ninguna medida por sí mismos ni la comunican, por ejemplo un tanque de agua del que se quiere conocer su nivel de llenado; también podría ser una habitación de un CPD de la que se quiere conocer su temperatura, humedad y otros parámetros ambientales. En estos casos es necesario sensorizar el objeto para poder medir datos, observar el comportamiento y enviarlo a través de Internet.

4. El cuarto tipo se usa cuando no existe el objeto como tal, es decir, se busca crear un prototipo de un futuro objeto. En esa situación, se crea el modelo 3D y se le envían datos para ver cómo actuaría en una situación real. En el Caso 2 se estudia este tipo de objetos.

Puede ocurrir que los sensores o los objeto únicamente tenga la capacidad de una conexión de corto alcance, como Bluetooth o 6LoWPAN, que imposibilite la comunicación con la plataforma. En esos casos, lo más recomendable es usar un *gateway* intermedio con capacidad para comunicarse con los sensores locales y con la plataforma alojada en un servidor

de Internet. Por ejemplo, la placa Creator Ci40 usada para los casos de estudio y que se explicara en la sección 2.3.2 es un buen caso de *gateway*.

3.3. Tipos de datos intercambiados

Los datos intercambiados entre objetos reales y modelos virtuales obedecen a distintos tipos que deben ser distinguidos por la aplicación.

Para organizar los datos se ha seguido a grandes rasgos la clasificación de variables de un artículo de *La Innovación Necesaria*[30]:

- **Puntuales**. Hacen referencia a eventos y acciones. Un evento ocurre cuando se interacciona con el modelo virtual, por ejemplo, se cambia un *switch*. El evento se manda siempre de plataforma a objeto. Y una acción es una orden que el objeto real da al modelo virtual, por ejemplo, levanta una tapa. La acción se manda siempre de objeto a plataforma. Los eventos no se guardan y las acciones –además de desencadenar una animación por parte del modelo 3D– pueden tomar cuatro estados que se guardan: START, PAUSE, RESUME y STOP. Los datos puntuales se guardan como valores de tipo «0».
- **Categóricos**. Se dividen en tres tipos:
  - **Dicotómicos**. Solo toman dos valores, que puede traducirse en 0 y 1. Por ejemplo, en una sala hay gente o no. Se guardan como valores de tipo «1».
  - **Nominales**. Son variables cuyos valores representan una categoría. Por ejemplo, el cuadrante que ocupa un objeto visto desde una cámara. Se guardan como valores de tipo «2».
  - **Ordinales**. Estas variables también representan una categoría pero se diferencian de las nominales en que se pueden ordenar. Por ejemplo, nivel de ruido bajo, medio o alto. Se guardan como valores de tipo «3».

Los valores de las categorías de los tres tipos están preestablecidos desde la creación del modelo.

- **Numéricos**. Se dividen en dos tipos:

  - **Discretos**. Son datos numéricos que toman valores contables entre un límite inferior y otro superior. Por ejemplo, el porcentaje de batería de un sensor. Se guardan como valores de tipo «4».
  - **Continuos**. Son datos numéricos que toman cualquier valor. Es importante conocer en qué unidades se mide. Por ejemplo, datos de temperatura. Se guardan como valores de tipo «5».

3.3.1. Comunicando los cambios

La comunicación de los cambios con la plataforma se realiza mediante el protocolo MQTT. En este protocolo, hay *topics* a los que un cliente se subscribe y sobre los que recibirá notificaciones. En concreto, en la plataforma se esperan dos *topics* fundamentalmente: medidas y acciones, y uno en los objetos: eventos. En la Figura 3.2 se observa que hay un *broker* central que distribuye los mensajes. Los clientes se tienen que subscribir a él con un *topic* específico, otros clientes (o el mismo) publican mensajes con un *topic* y los clientes que se han subscrito a ese *topic* reciben el mensaje. En este proyecto, la plataforma es el *broker*, pero también es un cliente, y cada objeto real es un cliente. El protocolo MQTT se explica más ampliamente en el Apéndice C.2.

Para publicar mensajes o subscribirse a ellos con medidas, acciones o eventos hay que usar el *topic*: dgiotwins/user/{username}/stage/{idEscenario}/data/{nombreDato}. Donde:

- **username** es el nombre elegido al crear un usuario.
- **idEscenario** es el identificador único que se le asigna a un escenario.
- **nombreDato** es el nombre dado a la acción, evento o medida.

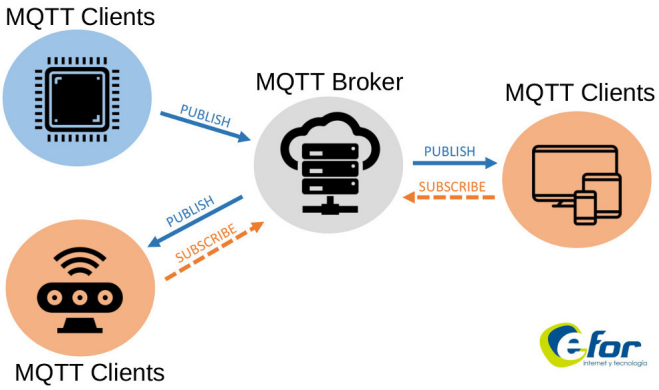


Figura 3.2: *Arquitectura MQTT [10].*

También se ha creado un protocolo propio que es obligatorio usar. El protocolo, que se muestra en la Figura 3.3, consta de 2 bytes de cabecera y los otros 254 bytes del mensaje. La cabecera a su vez se divide en 1 byte de operación en el que los 4 bits más significativos son fijos (0xD o 0b1101), el siguiente bit es un 0 si la comunicación va en sentido cliente servidor y un 1 si va en sentido servidor cliente; los últimos 3 bits del primer byte hacen referencia al tipo de dato: 0 (0b000) para puntual, 1 (0b001) para categórico dicotómico, 2 (0b010) para categórico nominal, 3 (0b011) para categórico ordinal, 4 (0b100) para numérico discreto y 5 (0b101) para numérico continuo, y el segundo byte es la longitud del mensaje. Si el lenguaje usado es C, cada byte de cabecera debe ser de tipo «uint_8» y el mensaje de tipo «char*». Para las medidas, el mensaje será simplemente la medida tomada, para los eventos el mensaje irá vacío y para las acciones el mensaje será 0 si se quiere arrancar la acción (START), 1 si se quiere pausar (PAUSE), 2 si se quiere reanudar (RESUME), 3 si se quiere parar (STOP) y 4 si se quieren parar todas las acciones del escenario (STOP_ALL).

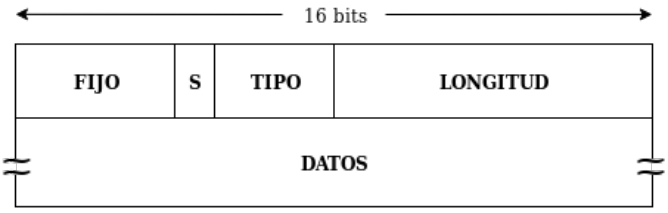


Figura 3.3: *Protocolo de comunicación creado para la plataforma.*

Capítulo 4

Requisitos funcionales

4.1. Introducción

En la sección 1.2 se han fijado los objetivos que la plataforma busca lograr. Siguiendo estos objetivos se pueden especificar varios requisitos que hay que cumplir: **medición y almacenamiento** de datos, **sincronización** de un objeto y su modelo; **visualización** de los objetos en la plataforma, **comunicación** entre objeto y plataforma, y, adicionalmente, **gestión** de la plataforma. Este capítulo detallará cada uno de los requisitos necesarios. Para definirlos, se usarán los términos «historias de usuario» y «*epic*» aludiendo a un conjunto de historias de usuario comunes según la metodología ágil de desarrollo de *software* Scrum [29].

La estructura de la plataforma se construye para realizar dos conceptos: modelo 3D y escenario. Un **modelo 3D** es un archivo que contiene la representación en tres dimensiones de un objeto real y puede tener varias animaciones establecidas. Representa de forma abstracta un objeto real, por ejemplo, una lámpara, pero no está asignada a ninguna lámpara real. Un **escenario** representa un objeto real en concreto y se sincroniza con él, por ejemplo, la lámpara del techo de un taller en particular. Para crear este escenario se necesita el modelo 3D de la lámpara, las acciones que puede llevar a cabo la lámpara del taller, los eventos que puede recibir y los datos que es capaz de medir. Cada escenario tiene un nombre único para poder sincronizarlo correctamente. De esta manera, con un único modelo «lámpara» se pueden representar varias lámparas distintas.

La dinámica de interacción entre el modelo virtual y el objeto físico se explicó en el Capítulo 3. Sin embargo, queda pendiente explicar cómo se debería usar la plataforma para crear el modelo virtual y para digitalizar el objeto físico. Para ello se plantean varios escenarios en los que se distinguen los siguientes actores principales:

- **objeto real**. El objeto físico al que se han fijado los sensores.
- **Medid**a. Datos medidos por los sensores de un objeto real.
- **Gemelo virtual**. El objeto virtual que representa al estático en la plataforma.
- **Plataforma**. Aloja uno o varios gemelos virtuales. Da acceso también al usuario.
- **Usuario**. Persona que utiliza la plataforma.

4.2. Medición y almacenamiento

Un gemelo digital de un objeto real no aporta absolutamente nada, si el objeto no es capaz de recolectar algún tipo de dato, ya sea medidas de cualquier tipo (temperatura, humedad...) o el propio estado del objeto (encendido/apagado, quieto/moviéndose...), no merece la pena virtualizarlo. O, por lo menos, no con el objetivo con el que se hace un gemelo digital.

Una vez recogidos los datos por el objeto, se deben enviar a la plataforma (ver sección 4.5) y se tienen que guardar.

Para esta *epic* hay dos roles: el objeto y la plataforma, y las historias de usuario son:

(H 4.2.1) Como objeto, quiero medir datos de mi entorno para obtener información. Esta información se compila y se envía a la plataforma. Si no hay comunicación, se pierden los datos.

Aceptación:

a) El objeto usa sensores para recoger datos ambientales como humedad, ruido, etc; propios como velocidad, temperatura, etc, o de estado como apagado/encendido, lleno/vacío, etc.

(H 4.2.2) Como plataforma, quiero almacenar los datos que recibo, para no perder nuevos datos que puedan llegar. Es un proceso continuo.

Aceptación:

a) La plataforma almacena en una base de datos los datos que recibe.

4.3. Sincronización

Todos los datos que son tomados en el objeto real, ya sean medidas de los sensores o el estado del objeto, se deben reflejar en el modelo 3D de la plataforma. Por lo que se asume que el objeto real y el gemelo comparten el mismo estado. Los cambios en uno y otro se sincronizan. Estos cambios se categorizan en **acciones** virtuales, **eventos** virtuales y **medidas** tomadas por los sensores. Los eventos virtuales representan algo que le ha sucedido al gemelo virtual y que debe trasladarse al objeto físico, se componen de alguno de los eventos HTML mencionados anteriormente (*click*, *mouse-in*, etc) disparados sobre una parte concreta del modelo y llevan asociados un nombre único. Por ejemplo, hacer *click* en un botón.

El resultado de una acción virtual es una composición de animaciones del modelo 3D que imitan una acción real sobre el objeto real, como levantar una tapa. Un objeto real podrá generar un número concreto de tipos de acciones. Cada acción generada se envía desde el objeto a la plataforma.

Algunos objetos estáticos son capaces de medir ciertos datos desde sus sensores que deben comunicar a la plataforma, por ejemplo, la temperatura ambiental.

Para esta *epic* hay tres roles: el objeto, su modelo y el usuario, y las historias de usuario son:

(H 4.3.1) El usuario realiza una modificación en el modelo 3D usando el ratón y este cambio debe reflejarse en el objeto real. El evento virtual se transmite al objeto real, donde se procesa y se concreta en un cambio sobre el objeto físico.

Aceptación:

a) Cuando un usuario interactúa con el gemelo virtual el objeto real recibe un mensaje con el evento que se ha disparado.

(H 4.3.2) El usuario toma el objeto real y realiza una acción sobre él, como pulsar un botón. Esto genera una acción virtual que se transmite al gemelo virtual. El gemelo la recibe y cambia de forma acorde usando una animación preprogramada.

Aceptación:

a) Cuando el objeto realiza una acción el modelo recibe un mensaje con la acción ejecutada.

b) El modelo virtual recibe la acción aunque esté apagado.

c) El modelo realiza la acción correspondiente.

d) No se paran o se descompasan las acciones al apagar y encender el modelo virtual.

4.4. Visualización

Para comprobar que el objeto real y su gemelo virtual están sincronizados, es necesario poder visualizar el modelo 3D del gemelo, que en él se puedan reflejar los cambios que sufre el objeto y que se pueda interactuar con el modelo 3D visualizado efectuando cambios también en el objeto real.

En la sección 2.2 se ha visto que no todos los *frameworks* optan por mostrar una copia en tres dimensiones del objeto. Sin embargo, desde un primer momento, en este proyecto se

ha considerado que era una parte importante visualizar los objetos en 3D. Aunque para el desarrollo de un gemelo digital este requisito no es indispensable, cumplir esta funcionalidad es lo que hace a este *framework* diferente a la mayoría de los que están en el mercado. Esto plantea nuevos retos: como recoger acciones y que partes del objeto ver.

No se puede olvidar que si el objeto puede hacer algún movimiento, el modelo también tiene que realizarlo, no es un modelo estático.

Aparte, se ha considerado que visualizar el objeto desde distintos dispositivos aporta más valor. Por lo que se muestra en el navegador del ordenador o *smartphone*, en realidad virtual y en realidad aumentada.

Aprovechando que se ha creado un módulo para visualizar los objetos, también se muestran distintas gráficas con todos los datos obtenidos del objeto real.

Para esta *epic* hay dos roles: la plataforma y el usuario, y las historias de usuario son:

(H 4.4.1) Un usuario accede a la plataforma, se identifica y quiere ver sus modelos 3D asignados. La plataforma muestra los objetos como modelos 3D para ayudar al usuario a obtener información sobre su objeto.

Aceptación:

a) El modelo se muestra en un navegador, en realidad virtual y en realidad aumentada.

b) El modelo solo se muestra a su usuario propietario.

c) El modelo realiza acciones visuales (animaciones) como girar unas hélices o encender un led.

d) El usuario realiza una acción sobre el modelo 3D (evento) y ve cambios en él.

Errores:

a) Si el modelo no existe mostrar un error 404.

b) Si el modelo no pertenece al usuario logueado mostrar un error 403.

(H 4.4.2) Como usuario, quiero revisar un sensor en concreto para ver qué pasa en una hora concreta. Para ello visualiza las gráficas del gemelo virtual. Estas gráficas contienen datos almacenados y que se generan en el objeto real.

Aceptación:

a) Se ha seleccionado una gráfica.

b) Cuando el modelo tiene datos almacenados se muestra esta gráfica con esos datos.

(H 4.4.3) Como usuario, quiero interactuar con el modelo 3D para realizar un cambio u operación. El usuario espera que el objeto real refleje ese cambio.

Aceptación:

a) Un usuario hace *click*, doble *click*, *mouse-in* o *mouse-out* sobre una parte del modelo[1].

b) La plataforma envía un evento al objeto real cuando se interacciona con el modelo.

c) El objeto real muestra el cambio.

Errores:

a) Si hay un error de comunicación se muestra el mensaje adecuado.

4.5. Comunicación

Si un objeto real no se puede comunicar con su gemelo digital –en este caso con la plataforma desarrollada– es imposible que el objeto real envíe las alteraciones que sufre, que

[1]Se acepta no poder hacerlo en realidad aumentada

ambos actúen de la misma manera y, por lo tanto, que estén sincronizados. Por lo que este es uno de los requisitos más importantes.

En una solución específica para un despliegue IoT en concreto, con unos sensores ya conocidos, sería mucho más fácil diseñar una solución a medida. La mejor solución, de poder hacerse, sería por cable y con unos mensajes prefijados. Sin embargo, la plataforma desarrollada en este proyecto busca ser una solución genérica para casi cualquier sensor, por lo que la solución tiene que ser lo más universal posible.

Como interfaz de comunicación se puede usar cualquiera: WiFi, Bluetooth, 6LoWPAN, ethernet, etc. El protocolo de comunicación usado es MQTT, ya que permite crear *topics* jerarquizados y facilita la tarea. MQTT se explica más ampliamente en el Apéndice C.2. La comunicación es bajo demanda para ahorrar energía. Solo se comunica cuando hace falta. Además, esta comunicación sigue un protocolo definido en la sección 3.3.1.

Para esta *epic* hay dos roles: el objeto y la plataforma, y las historias de usuario son:

(H 4.5.1) Como objeto real, quiero comunicarme con la plataforma para enviar datos de medidas y acciones realizadas sobre mí, y recibir datos de eventos generados por mi gemelo virtual.

Aceptación:

a) El objeto se conecta a la red a través del protocolo elegido, envía los datos y la plataforma los recibe.

b) El objeto escucha a la espera de nuevos datos.

(H 4.5.2) Como plataforma, quiero comunicarme con cualquier objeto para enviar datos de eventos sobre el gemelo virtual y recibir datos de medidas y acciones.

Aceptación:

a) La plataforma se conecta a la red a través del protocolo elegido, envía los datos y el objeto adecuado los recibe.

b) La plataforma escucha a la espera de nuevos datos.

4.6. Gestión

Es necesario facilitar subir modelos, editarlos, gestionarlos e identificarlos para poder sincronizarlos con un objeto real.

Hay un único rol: el usuario, y las historias de usuario son:

(H 4.6.1) El usuario ha creado una malla 3D en uno de los formatos permitidos. Quiere subirla y asignarle un nombre para completar la definición del escenario para el gemelo virtual.

Aceptación:

a) Aparece un «modal» para subir un archivo con la malla 3D y asignar un nombre.

b) Los parámetros del modelo 3D (ver sección 5.3.1.4) se guarda en la base de datos con el nombre asignado.

c) El archivo que contiene la malla 3D se almacena en el disco duro del servidor.

Errores:

a) Si el nombre ya está asignado a algún modelo del mismo usuario devuelve un error.

b) Si el nombre incluye caracteres no válidos devuelve un error.

c) Si no se puede subir devolver error 500.

(H 4.6.2) Como usuario, quiero crear un escenario a partir de un modelo 3D para virtualizar un objeto real.

Aceptación:

a) Se le puede poner un nombre genérico.

b) Se le puede asignar un identificador único.

c) Se le puede añadir un modelo entre todos los del usuario.

d) Se puede adornar opcionalmente con un paisaje. Hay dos tipos de paisaje: de una sola imagen, en este caso aparecerá el modelo sobre una imagen de fondo y al rotarlo no cambia el paisaje, y de seis imágenes, en este caso se sitúa el objeto como si estuviese dentro de un cubo donde cada cara del cubo tiene una de las imágenes y al rotarlo va variando el paisaje. De la segunda forma es mucho más realista.

e) Se le pueden configurar acciones virtuales, eventos virtuales y medidas.

f) El escenario se guarda en la base de datos.

Errores:

a) Si el identificador único ya está en uso devuelve un error.

b) Si el identificador incluye caracteres no válidos devuelve un error.

(H 4.6.3) Como usuario, quiero ver una lista con todos mis modelos para incluir en el escenario actual.

Aceptación:

a) Se muestra una lista con todos los modelos subidos por un usuario.

b) Si se pincha en uno se accede a su información en concreto.

c) No aparecen modelos de otros usuarios.

d) No aparecen modelos ya borrados.

(H 4.6.4) Como usuario, quiero ver los detalles de un único modelo para conocer su información.

Aceptación:

a) Hay una pantalla donde aparece el modelo 3D con toda su información.

b) En esta pantalla se podrá elegir cómo ver el modelo: en el navegador, en realidad virtual o en realidad aumentada.

c) Si tiene animaciones asociadas aparecerá un botón por cada animación que las pondrá en marcha.

Errores:

a) Si se intenta acceder a un modelo que no existe se mostrará un error 404.

b) Si se intenta acceder a un modelo de otro usuario se mostrará un error 403.

(H 4.6.5) Como usuario, quiero cambiar parámetros del modelo (rotación y escala) para que se adapte más a la realidad.

Aceptación:

a) Existe un botón de editar modelo.

b) Al pulsar el botón «editar» aparece un panel donde se puede disminuir o aumentar el tamaño y la rotación en cada uno de los ejes.

c) Al editar el modelo se guarda en la base de datos y la siguiente vez que se acceda mantendrá los nuevos parámetros.

(H 4.6.6) Como usuario, quiero borrar un modelo.

Aceptación:

a) Hay un botón de borrar modelo.

b) Al pulsar el botón pide confirmación para borrarlo.

c) Al borrarlo desaparece de la base de datos y se elimina el fichero del disco duro.

d) Una vez borrado no aparece el modelo en la lista de modelos del usuario.

e) Una vez borrado se puede volver a usar el nombre.

Errores:

a) Si hay un error al borrar se devuelve un error 500.

(H 4.6.7) Como usuario, quiero, al crear un escenario, configurar las acciones virtuales que puede realizar para sincronizarlas con las del objeto real. Para crear una acción virtual es necesario identificar animaciones a asociar que pertenecen al modelo 3D del escenario. Se espera que el objeto real, cuando se de esa acción en él, la transmita al virtual.

Aceptación:

a) Cada acción debe tener un nombre único dentro del escenario.

b) En cada acción virtual se pueden combinar varios animaciones.

c) En una acción no puede haber dos repeticiones iguales.

d) Cada animación se puede configurar con el número de repeticiones (o 0 para repeticiones infinitas).

e) En cada animación se puede configurar en qué posición acaba: en el último *frame* o vuelve al primero. En caso de repeticiones infinitas esta opción es irrelevante.

f) Antes de crear el escenario se pueden borrar las acciones ya configuradas.

Errores:

a) Si se intenta poner un nombre ya existente devuelve un error.

b) Si se deja algún campo vacío devuelve un error.

c) Si se intenta añadir una animación que ya está añadida devuelve un error.

(H 4.6.8) Como usuario, quiero, al crear un escenario, configurar los eventos virtuales que puede disparar para lanzarlos al objeto real. Los posibles eventos HTML son: *click*, doble *click*, *mouse-in* o *mouse-out*.

Aceptación:

a) Cada evento debe tener un nombre único dentro del escenario.

b) A cada evento del escenario se le puede asociar un (y solo un) evento HTML.

c) A cada evento se le puede asociar las partes del modelo donde se puede leer el evento. Por ejemplo, hacer *click* sobre un botón o *mouse-in* sobre un sensor de movimiento.

d) Antes de crear el escenario se pueden borrar los eventos ya configurados.

Errores:

a) Si se intenta poner un nombre ya existente devuelve un error.

b) Si se deja algún campo vacío devuelve un error.

(H 4.6.9) Como usuario, quiero, al crear un escenario, configurar los datos de medidas que puede tomar para que coincidan con las del objeto real.

Aceptación:

a) Cada medida debe tener un nombre único dentro del escenario.

b) Se puede elegir el tipo de dato: categórico nominal, categórico dicotómico, numérico discreto o numérico continuo.

c) Una vez elegido el tipo de dato aparecerá un nuevo campo que dependerá del tipo de dato elegido:

- Categórico nominal o dicotómico: aparecerá un campo de texto donde se introducirán las posibles categorías.
- Numéricos discretos: aparecerán dos campos numéricos con los valores mínimo y máximo que se pueden medir.
- Numéricos continuos: aparecerá un campo de texto para introducir las unidades en las que se miden los datos.

d) Antes de crear el escenario se pueden borrar las medidas ya configuradas.

Errores:

a) Si se intenta poner un nombre ya existente devuelve un error.

b) Si se deja algún campo vacío devuelve un error.

(H 4.6.10) Como usuario, quiero ver una lista con todos mis escenarios para trabajar con ellos.

Aceptación:

a) Se muestra una lista con todos los escenarios creados por un usuario.

b) Si se pincha en uno se accede a su información en concreto.

c) No aparecen escenarios de otros usuarios.

d) No aparecen escenarios ya borrados.

(H 4.6.11) Como usuario, quiero ver los detalles de un único escenario para conocer su información.

Aceptación:

a) Hay una pantalla donde aparece un escenario con toda su información.

b) En esta pantalla se podrá cambiar entre modos de vista: navegador, realidad virtual y realidad aumentada.

c) Si tiene un paisaje, se verá el modelo sobre el paisaje.

d) En esta pantalla el modelo realizará las acciones cuando corresponda, es decir, cuando las realice el objeto real.

e) Desde esta pantalla se podrá interactuar con el modelo para generar eventos.

f) En esta pantalla aparecerán gráficas con los datos medidos por el objeto real que se representa.

Errores:

a) Si se intenta acceder a un escenario que no existe se mostrará un error 404.

b) Si se intenta acceder a un escenario de otro usuario se mostrará un error 403.

(H 4.6.12) Como usuario, quiero borrar un escenario.

Aceptación:

a) Hay un botón de borrar escenario.

b) Al pulsar el botón pide confirmación para borrarlo.

c) Al borrarlo desaparece de la base de datos.

d) Una vez borrado no aparece el escenario en la lista de escenarios del usuario.

e) Una vez borrado se puede volver a usar el nombre.

Errores:

a) Si hay un error al borrar se devuelve un error 500.

Se puede observar que algunos de los requisitos están ligados entre sí. Por ejemplo, un objeto y su modelo no pueden estar sincronizados si el objeto y la plataforma no se pueden comunicar.

En el siguiente capítulo de arquitectura se detallará cómo se ha llevado a cabo las soluciones propuestas.

Capítulo 5

Arquitectura

5.1. Introducción

En este capítulo se explica la arquitectura de la plataforma diseñada para cumplir los requisitos marcados en el Capítulo 4. Se empezará presentando una visión general de la arquitectura, identificando el funcionamiento del servidor y la conexión con el mundo físico. Entre tanto, se detallarán los cometidos de cada módulo y submódulo; la comunicación de los datos y su manejo.

La comunicación entre componentes se realiza a través de una API REST y la implementación con NodeJS. Siguiendo con esta idea, para cada componente se enumerarán los servicios prestados y se detalla la operación (GET, POST, PUT o DELETE) utilizada.

5.2. Elementos arquitectónicos

En el proyecto se pueden diferenciar dos módulos completamente separados pero conectados entre sí a través de MQTT como muestra la Figura 5.1:

- **Servidor principal:** es el módulo central del proyecto. Es la plataforma donde los usuarios pueden subir, gestionar y visualizar sus gemelos digitales de objetos reales. Aquí se alojan los modelos virtuales. A su vez está divido en varios submódulos:
  - Administrador de modelos 3D. Gestiona todo lo que tiene relación con los modelos de objetos: la subida, el almacenamiento y la visualización en varios formatos.

- Administrador de escenarios. Se encarga de crear, almacenar y hacer que funcionen los escenarios. Se compone de dos partes complementarias:
  - Modelo de datos. Cada escenario guarda las medidas tomadas por los sensores del objeto.
  - Modelo 3D. Cada escenario representa un objeto real en concreto y necesita un modelo 3D. Hace uso del módulo anterior.

- **Mundo físico:** hace referencia al mundo real. Aquí se encuentran los objetos reales que son sensorizados y luego se representan en la plataforma. Este objeto real debe ser capaz de recolectar datos y enviarlos a través de uno o varios sensores que se asume que tiene el objeto.

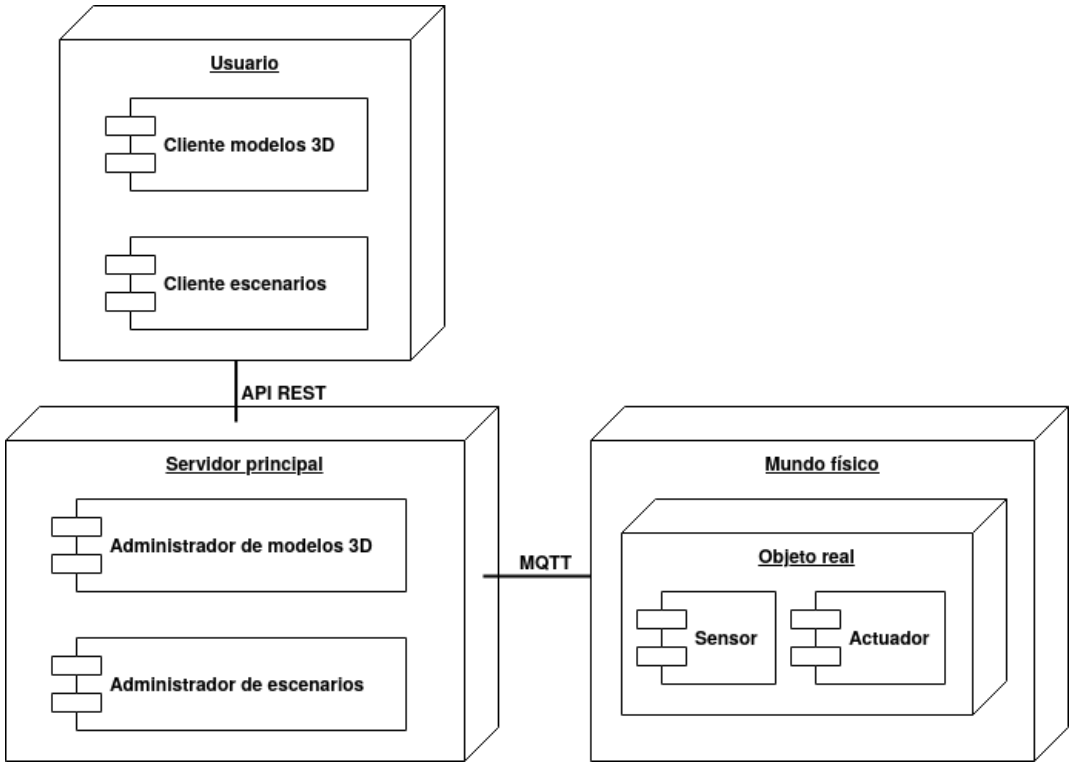


Figura 5.1: *Arquitectura del framework. Se pueden diferenciar los módulos de la arquitectura y la comunicación entre ellos. Dentro del servidor aparecen los Administradores de modelos y escenarios.*

5.3. Servidor principal

Su tarea principal es establecer la comunicación con el mundo físico para intercambiar y recolectar los datos entrantes para plasmarlos en el modelo virtual. Provee al usuario de la capacidad para crear el modelo virtual, vincular partes de él a datos del objeto real, interactuar con el modelo y enviar datos del modelo al objeto real.

Se divide en los dos submódulos que aparecen en la Figura 5.2: el Administrador de **modelos 3D** crea, edita y gestiona las representaciones virtuales de los objetos reales, y el Administrador de **escenarios** hace uso de los modelos para crear un escenario único que recibe datos de medidas y acciones de un objeto real, y le envía eventos.

Al dividirse en estos dos submódulos permite reutilizar un mismo modelo 3D en diferentes escenarios. Por ejemplo, para «clonar» un parque eólico se puede crear un único modelo 3D de una aerogenerador y diferentes escenarios para cada aerogenerador real con sus propios datos.

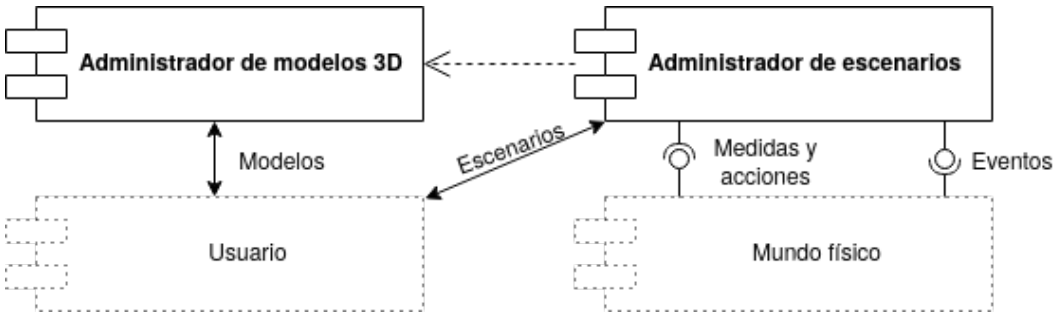


Figura 5.2: *Interconexión módulos del servidor principal. El Administrador de modelos recibe una representación virtual de un objeto real, el Administrador de escenarios usa esta representación y se comunica con el mundo físico, el servidor le envía eventos al objeto y el objeto envía datos de medidas y acciones.*

5.3.1. Administrador de modelos 3D

El Administrador de modelos 3D consta de cuatro submódulos que se encargan de que un usuario pueda subir, editar, gestionar y ver sus modelos: el **Editor** de parámetros de los modelos 3D, el **Controlador** de modelos 3D, el **Visualizador** de modelos 3D y el

Repositorio de modelos 3D. Se interconectan como muestra la Figura 5.3.

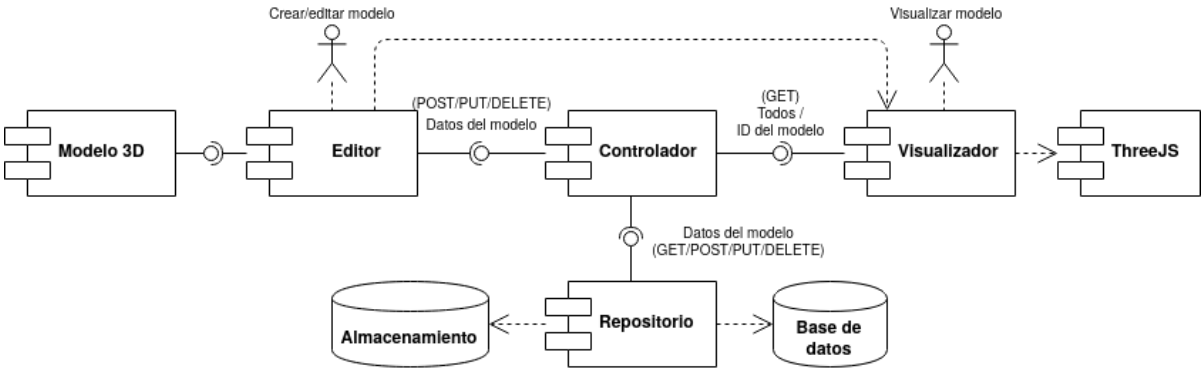


Figura 5.3: *Diagrama de componentes del Administrador de modelos 3D.*

El Editor y el Visualizador se encuentran en la parte de cliente, y el Controlador y el Repositorio en la parte de servidor. Desde el Visualizador, el usuario puede ver una lista con todos los modelos y cada modelo de forma individual. El Visualizador, de forma interna, se comunica con el Controlador para pedir los modelos correspondientes. Cuando un usuario visualiza un único modelo tiene la opción de modificar sus datos. El Editor recoge los datos modificados y se los envía al Controlador. El Editor, además, crea modelos nuevos a partir de un fichero de objeto 3D.

El Controlador gestiona todos los datos de los modelos, cuando le llegan nuevos datos se los envía al Repositorio para guardarlos y cuando le piden datos de un modelo los proporciona. El Repositorio ofrece persistencia a los modelos, almacena en el disco duro los ficheros 3D y guarda en una base de datos los valores del modelo (Los valores se explican en la sección 5.3.1.4).

Usa un patrón *Model View Controler* (MVC) donde el Controlador está implementado en NodeJS, la vista –el Editor y el Visualizador– en HTML, CSS y JS, y el modelo está representado en el Repositorio con una base de datos MongoDB.

5.3.1.1. Controlador de modelos 3D

El Controlador ofrece servicios REST al cliente (ver Figura 5.3). Desarrolla dos cometidos principales:

1. La primera es crear, modificar y eliminar un modelo. Cuando un usuario sube un modelo, el Controlador comprueba que no supera el tamaño máximo (10 MB) y que ningún modelo del usuario se llama igual. Si todo es correcto, guarda el fichero en un directorio del servidor y los datos del modelo (Se explican en la sección 5.3.1.4) en la base de datos, donde la escala inicial se fija a «1» y la rotación en todos los ejes a «0».

   Cuando un usuario quiere modificar un modelo debe pasar el nombre del modelo y los nuevos datos para actualizar el modelo. El Controlador, cuando le llegan los datos, comprueba que el modelo pertenece al usuario y si es correcto lo actualiza con los nuevos datos.

   Para borrar un modelo, el usuario debe pasar el nombre del modelo. Y si el modelo pertenece al usuario borra el documento de la base de datos y el fichero del servidor. En la Figura 5.4 se muestra un diagrama de flujo de estas tres funciones.

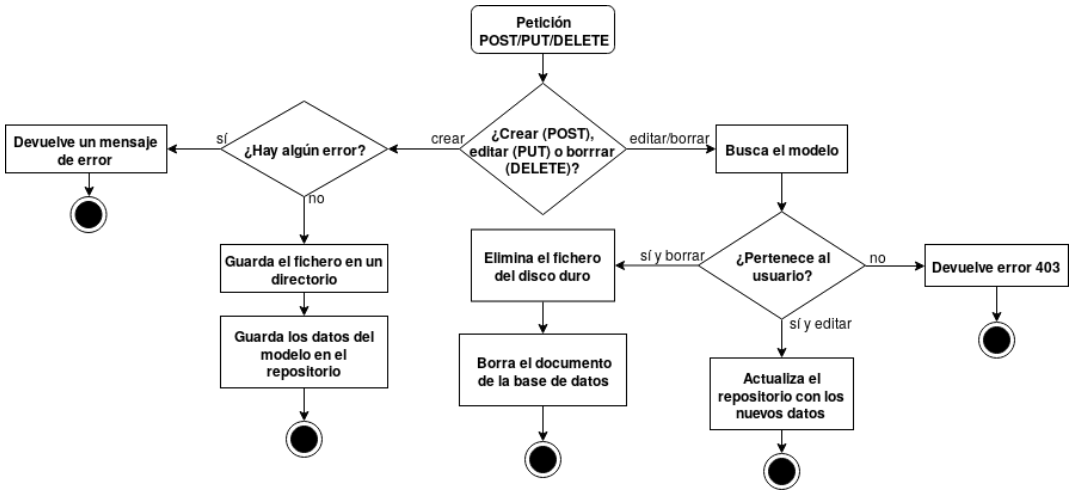


Figura 5.4: *Diagrama de flujo que explica los pasos para crear, modificar y borrar un modelo.*

2. La segunda función es proveer al usuario de los modelos que pide. Un usuario puede

pedir la lista con todos sus modelos o un modelo en particular. En el caso de pedir todos los modelos, el Controlador los busca en la base de datos y envía al usuario los datos de los modelos junto con sus ficheros 3D correspondientes. Si no hay ningún modelo cuyo propietario sea el usuario, envía una lista vacía. Si el usuario pide un modelo en concreto, lo primero que hace el Controlador es comprobar que le pertenezca, si no es así envía un mensaje de error. Si todo es correcto, le envía el modelo y su fichero 3D. En la Figura 5.5 se muestra un diagrama de flujo de este proceso.

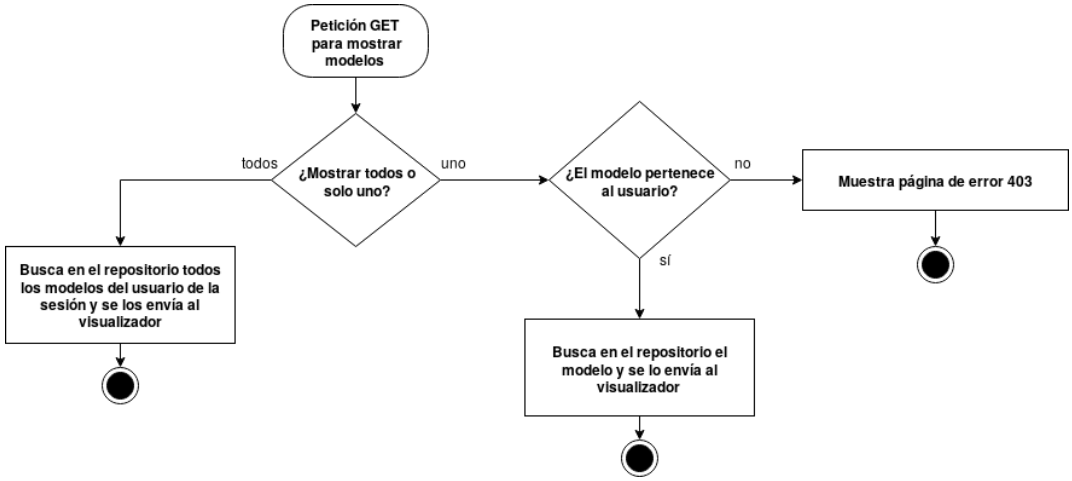


Figura 5.5: *Diagrama de flujo que explica los pasos desde que un usuario pide un modelo hasta que el Controlador lo devuelve.*

5.3.1.2. Visualizador de modelos 3D

El Visualizador proporciona una vista donde el usuario puede ver una lista con todos sus modelos y cada uno de ellos con más detalles. La parte de la interfaz gráfica se explicará en la sección 6.2.1, este capítulo se centra en cómo se muestra.

En la Figura 5.6 se muestra la secuencia que se sigue para mostrar un modelo 3D en la interfaz del Visualizador.

La biblioteca ThreeJS se encarga de renderizar el modelo 3D para poder mostrarlo. Para conseguirlo, el primer paso es «iniciar» el fichero con el objeto 3D y el segundo es

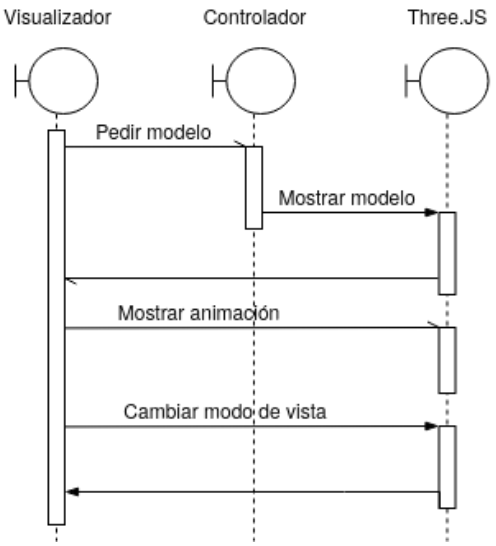


Figura 5.6: *Diagrama de secuencia que explica cómo se muestra un modelo.*

«animarlo». Ejecuta dos funciones principales: *init* y *animate*, y dos secundarias: *createGUI* y *render*. La Figura 5.7 muestra esta secuencia.

La función ***init*** empieza localizando la zona de la página web donde se dibujará el modelo, crea una instancia de **reloj** que sirve para traquear el tiempo en la animaciones y crea una **escena** vacía. Después le va añadiendo ítem a la escena: el **elemento** exacto donde aparecerá el modelo, la **cámara** desde donde se «ve» la escena, esta cámara hay que configurarla con el tipo, el campo de visión, el ratio, los puntos de máximos de acercamiento y alejamiento; la posición y la dirección de enfoque, y los **controles**, también hay que configurar su tipo e indicarle cuales con la cámara y el elemento. A continuación lee la extensión del modelo para poder utilizar el ***loader*** correspondiente y lo ejecuta. Al ejecutar el *loader* descarga el fichero, si tiene animaciones crea –de forma asíncrona– el cuadro de mostrar animaciones y las configura (función *createGUI*); actualiza la escala y la rotación del objeto con los datos guardados en la base de datos, y añade a la escena el objeto descargado, **luz** y una cuadrícula para que no de la sensación de estar flotando. Mientras esté paso está en proceso muestra una barra de progreso. Por último, crea la instancia que será utilizada para renderizar el objeto. Una vez iniciado el objeto 3D, llama a la función *animate*.

La función ***animate*** lee, a través del reloj creado anteriormente, el tiempo que ha pasado desde la última vez que se ejecutó está función (o inicia el reloj si es la primera vez), si hay animaciones configuradas «avisa» de que ya ha pasado el tiempo correspondiente para que, en el caso de que haya una animación ejecutándose, actualice la posición del objeto; llama a la función *render* para renderizar el objeto con su nueva posición y se llama a sí misma. Por lo que esta función se ejecuta en bucle hasta que el usuario cierre la página.

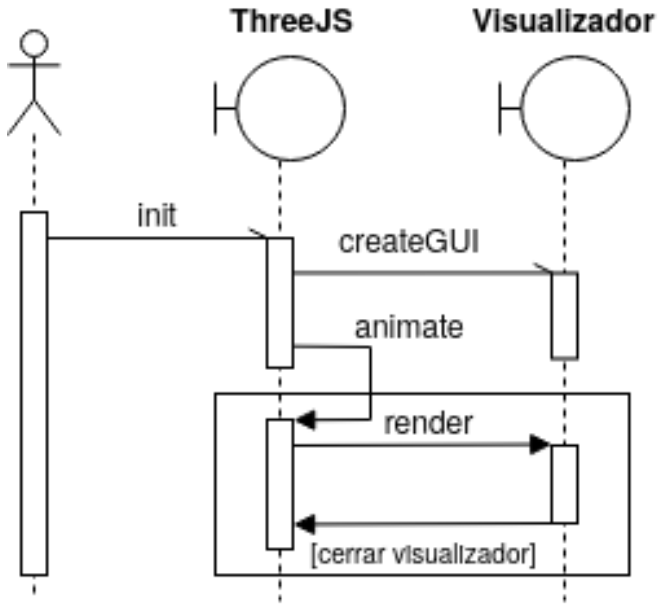


Figura 5.7: *Secuencia de renderización de un modelo.*

La función ***createGUI*** crea una interfaz gráfica con una lista de las animaciones del objeto. Si no tiene animaciones, en vez de crearla vacía, directamente no la crea. Cada animación de la lista es un botón con un *callback* que muestra la animación en el modelo y, en caso de que se esté reproduciendo otra animación, la desvanece para que se reproduzca la nueva. Cada animación se configura para que se ejecute un sola vez y para que una vez acabada, el modelo vuelva a su posición de origen.

La función ***render*** lee la posición de la cámara, el ancho y alto del contenedor donde se muestra el modelo, y la interacción del usuario con el modelo para fijar cómo se va a mostrar, y, finalmente, renderiza el objeto.

5.3.1.3. Editor de parámetros de los modelos 3D

El Editor aporta una interfaz gráfica para subir nuevos modelos y para editar los ya existentes. Su objetivo es que el usuario pueda comunicar los datos al Controlador de una forma sencilla.

Para subir un modelo el usuario únicamente deberá elegir un fichero 3D de su ordenador (en los formatos válidos y sin superar el tamaño máximo) y asignarle un nombre que no esté en uso.

Una vez subido, del modelo se puede editar su escala y la rotación en cada uno de sus ejes. La escala varía en un 10 % en cada cambio y la rotación en 0,1 radianes. También permite borrar el modelo enviando al Controlador el id del modelo a borrar. La Figura 5.8 muestra un diagrama de secuencia con la comunicación entre el Editor de parámetros y el Controlador.

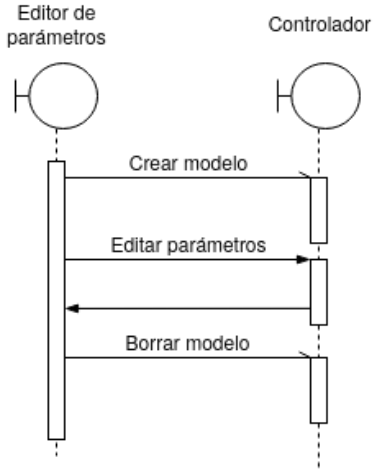


Figura 5.8: *Diagrama de secuencia del Editor de parámetros 3D.*

En la sección 6.2.2 se mostrará cómo es la interfaz gráfica que permite estos cambios.

5.3.1.4. Repositorio de modelos 3D

Todos los modelos 3D se deben almacenar en una base de datos para adquirir persistencia. En este proyecto se usa una base de datos MongoDB[1], que es una base de datos NoSQL de código abierto orientada a documentos. Los documentos se guardan en formato BSON, que es una representación binaria de JSON. Una de las ventajas de estas bases de datos es que los documentos de una misma colección –equivalente a una tabla en SQL– pueden tener esquemas diferentes con diferentes campos para cada documento. Cada documento tiende a tener toda la información necesaria, mientras que en una base de datos relacional, la información se divide en diferentes tablas. Gracias a esto, se convierten en bases de datos muy ágiles.

El Repositorio está formado por una colección llamada «models» que sigue el siguiente esquema:

- **_id**: Valor de tipo *ObjectId* alfanumérico. Es único para cada dato en una base de datos. Se asigna automáticamente. Solamente se usa de manera interna.
- **date_created**: Fecha de creación del modelo. Valor de tipo *ISODate*.
- **date_updated**: Fecha de la última modificación del modelo. Se inicia con la fecha de creación. Valor de tipo *ISODate*.
- **ext**: Es la extensión del fichero subido. Valor de tipo *string*.
- **name**: Nombre dado al modelo 3D. Valor de tipo *string*.
- **owner_id**: Identificador del usuario que ha subido el modelo. Valor de tipo *string*.
- **path**: Dirección donde se guarda el fichero. El nombre se forma con un combinación del ID del usuario que ha subido el modelo y del nombre del modelo. Valor de tipo *string*.

[1] http://www.mongodb.com

- **rotation**: Número de radianes que hay que girar el modelo antes de mostrarlo. Tiene un valor por cada eje: 'x', 'y' y 'z'. Valores de tipo numérico.
- **scale**: Número por el que hay que multiplicar la escala original del modelo antes de mostrarlo. Tiene un valor por cada eje: 'x', 'y' y 'z'. Valores de tipo numérico.

5.3.2. Administrador de escenarios

El proyecto requiere representar de forma digital un escenario que se da en la vida real, como puede ser un parque eólico, la maquinaria de una fábrica o un rover en Marte. El Administrador de escenarios se encarga de que un usuario pueda crear, editar y gestionar los escenarios; visualizar de forma digital el escenario viendo lo que ocurre en el real e interactuar con el escenario digital de la misma forma que se haría con el real. Este módulo está formador por cuatro submódulos: el **Editor** de escenarios, el **Controlador** de escenarios, el **Visualizador** de escenarios y el **Repositorio** de escenarios. Cada escenario está formado por un **Modelo 3D** y un **Modelo de datos**. Se interconectan como muestra la Figura 5.9. Donde también se ve que el Controlador es la conexión con el mundo físico y que el Modelo 3D del escenario depende de los modelos 3D que hay en el Administrador de modelos 3D.

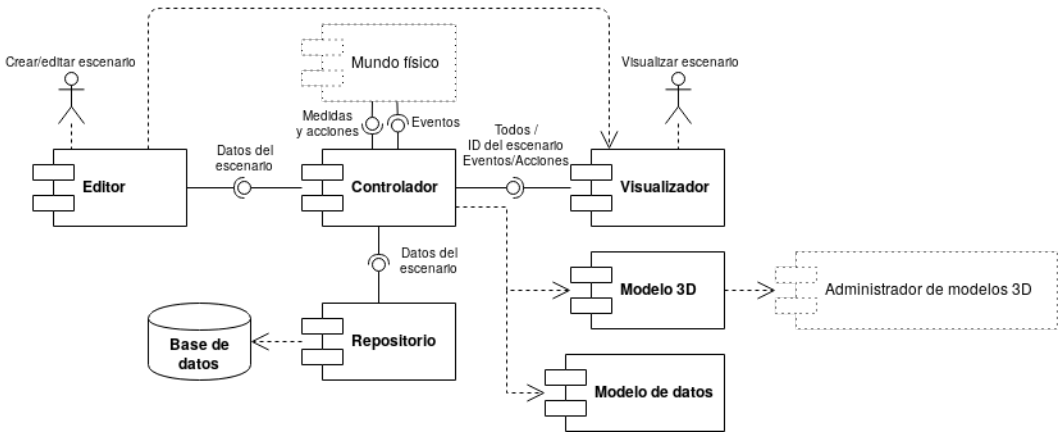


Figura 5.9: *Diagrama de componentes del Administrador de escenarios.*

La arquitectura sigue el mismo patrón que en el Administrador de modelos 3D con algunas ligeras diferencias. Los escenarios se crean a partir de un modelo 3D ya existente

y de un modelo de datos que incluye acciones, eventos y medidas tomadas por el objeto real. El Controlador, además de gestionar los escenarios en la plataforma, se comunica con el mundo físico para recibir acciones y enviar los eventos. Los eventos se generan desde el Visualizador y se envían al Controlador, y las acciones y las medidas llegan desde el objeto y el Controlador se las envía al Visualizador para que reproduzca la acción y muestre los datos en gráficas. Los valores que se guardan en la base de datos se explican en la sección 5.3.2.6.

5.3.2.1. Modelo de datos

El modelo de datos recoge cada uno de los datos generados por el escenario y que se envían a la plataforma. Estos datos pueden ser medidas realizadas, como la temperatura, una acción que ha realizado, como encender el led derecho, o un evento, como pulsar un botón.

Solo se admiten una serie de tipos de datos (ver sección 3.3): puntuales, categóricos dicotómicos, categóricos nominales, categóricos ordinales, numéricos discretos y numéricos continuos.

Para manejar y mostrar todos los datos de un escenario se hace uso de un **Controlador** de datos y un **Visualizador** de datos que forman parte del Controlador y Visualizador de escenarios respectivamente; por lo que se explicarán más adelante en sus correspondientes secciones.

5.3.2.2. Modelo 3D

El modelo 3D es la representación del objeto real que forma parte del escenario. Este modelo se elige entre los disponibles del Administrador de modelos 3D, por lo que la manera de visualizarlo y manejarlo viene dada por ese módulo, y está guardado en el Repositorio de modelos 3D (ver sección 5.3.1).

Sin embargo, para decir que es un «gemelo», tanto el objeto real como el virtual deben actuar de la misma manera. Para conseguir este objetivo en el modelo 3D se representan las mismas **acciones** que realiza el objeto real y comunicar los **eventos** que ocurren en el

modelo 3D al objeto real. Las acciones se muestran gracias a las animaciones que el modelo 3D tiene configuradas y para poder interactuar con el modelo y desencadenar un evento se usan técnicas de *ray casting* [31]. Cuando el modelo se está observado en modo de realidad aumentada no se puede interactuar con él.

5.3.2.3. Editor de escenarios

La función es idéntica al Editor de parámetros de modelos; aporta, a través del Visualizador, una interfaz gráfica para que un usuario pueda crear sus propios escenarios fijando sus parámetros y editar los escenarios ya existentes. Una vez recogidos los datos se los envía al Controlador y los guarda en el Repositorio. Su objetivo principal es facilitar la comunicación de los datos al Controlador.

Un escenario tiene el formato que se muestra en la Figura 5.10, cuyos datos se volverán a ver en la sección 5.3.2.6 del Repositorio.

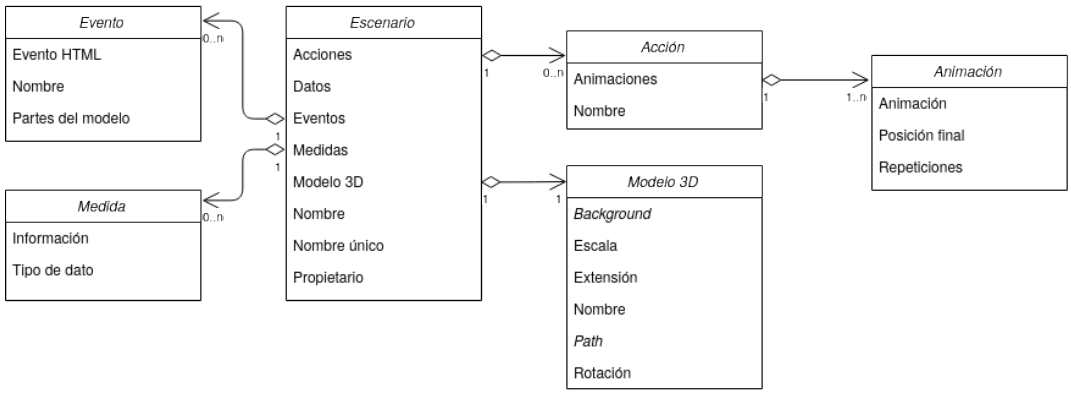


Figura 5.10: *Diagrama de clases de un escenario.*

Entre otras cosas, este Editor se encarga de crear las acciones de un escenarios, es importante diferenciar animación y acción: una animación es un movimiento que tiene el modelo 3D y una acción es la combinación de una o más animaciones a las que se les ha configurado de una manera concreta y están asociadas a un escenario.

Por ejemplo, hay un modelo de un helicóptero que tiene una animación que rota la hélice vertical y otra que rota la hélice horizontal, se puede crear una acción que les añada

la opción de ejecutarse indefinidamente a cada una y las combine para formar la acción «volar». En la Figura 5.11 se muestra una simplificación de la creación de una acción a partir de animaciones.

Figura 5.11: *Esquema simplificado de creación de una acción.*

La interfaz del Editor se explica en el Capítulo *Interfaz de usuario* en la sección 6.3.1.

5.3.2.4. Visualizador de escenarios

El Visualizador de escenarios puede mostrar una lista con todos o uno individual. El Visualizador individual se compone de dos Visualizadores distintos, en uno se ve el modelo 3D y en el otro, el Visualizador de datos, se ven gráficas con datos enviados por el objeto real. Cada vez que se abre el Visualizador individual se crea un canal de comunicación WebSockets con el Controlador para poder recibir datos de acciones y nuevas medidas. Su interfaz se mostrará en la sección 6.3.2, este capítulo se centra en cómo se visualiza.

Para visualizar el modelo, hace uso de las mismas funciones *init*, *animate* y *render* del Visualizador de modelos (Sección 5.3.1.2). Aunque la función *init* realiza algunas tareas diferentes.

Las diferencias entre esta función *init* y la anterior, es que esta no configura la animaciones asociadas al objeto, es decir, no ejecuta la función *createGUI*. En vez de añadir directamente una cuadrícula, primero comprueba si el escenario tiene paisaje, si tiene y es del tipo «*cube*», carga una textura de cubo con las seis imágenes del cubo, si tiene y es del tipo «*texture*», carga una textura normal con la imagen y si no tiene, añade la cuadrícula.

Si el escenario tiene aciones o eventos, prepara el escenario para la posible llegada de estas acciones y eventos, es decir, ejecuta las funciones *setupActions* y *setupEvents* respectivamente. Por último, si el escenario tiene datos que se puedan mostrar en gráficas los muestra, es decir, ejecuta la función *showData*.

La función ***setupActions*** combina todas las animaciones 3D de una acción en una sola animación 3D. Antes de combinarlas, configura las animaciones con los parámetros elegidos por el usuario. Para cada acción que tiene el escenario crea una función JS asociada al nombre de la acción que se ejecutará al iniciar el Visualizador y cuando el Controlador lo comunique a través de WebSockets. Esta función admite dos parámetros: el estado de la acción (START, PAUSE, RESUME y STOP) y el tiempo que lleva la acción iniciada. Si el objeto real manda una acción y el Visualizador no está abierto, no hay canal de comunicación y por tanto no ejecuta la acción en ese momento, por eso es importante saber el tiempo que la acción lleva iniciada para sincronizarla con el objeto real. Utiliza el estado para reiniciar la acción, pausarla, reanudarla o pararla del todo.

La función ***setupEvents*** lee los eventos virtuales del escenario y sobre qué partes del modelo se ejecutan, y crea un objeto JSON con los cuatro eventos HTML posibles («*click*», «*dblClick*», «*mouseIn*» y «*mouseOut*»). Cada uno de los eventos es otro objeto JSON donde la clave es la parte del modelo sobre la que se ejecuta el evento virtual y el valor es el nombre del evento virtual, como se ve en el siguiente ejemplo:

```
{
  "click": {
    "button_der": "click_btn_der",
    "button_izq": "click_btn_izq"
  },
  "mouseIn": {
    "sensor": "presence_detected"
  }
}
```

La siguiente tarea para que funcione es preparar al elemento para que escuche los eventos HTML «*onMouseDown*», «*onDblClick*» y «*onMouseMove*», y les asigna diferentes funciones

Javascript que se dispararán cuando se produzca el evento. Estas funciones traducen el evento que ha ocurrido a uno de los cuatro anteriores, leen en qué parte del modelo 3D se ha producido el evento a través de *ray casting* [31] y envían al Controlador el nombre del evento del escenario haciendo uso del objeto JSON formado anteriormente. Cada función tiene sus particularidades:

La función para ***onMouseDown*** tiene un antirrebotes para que cuando el usuario haga doble *click* no cuente también como dos *clicks*. El antirrebotes consiste en añadir un temporizador de 200 milisegundos a la espera de que se vuelva a hacer *onMouseDown* u *onDblClick*, si no se hace se envía el evento *click* y si se hace se cancela.

La función para ***onDblClick*** cancela un posible evento *click* y envía un evento *dblClick*.

La función para ***onMouseMove*** guarda en todo momento la parte del objeto en la que está ocurriendo y la última parte en la que ha ocurrido. Si la parte en la que está ocurriendo es diferente a la última en la que ocurrió, manda el evento *moveOut* para la última y *moveIn* para la nueva. Si no está ocurriendo sobre ninguna parte del modelo pero está guardada la última parte en la que ocurrió, manda el evento *moveOut* para la última parte guardada. Y si está ocurriendo sobre una parte del modelo pero no hay guardada ninguna anterior, manda el evento *moveIn* para la nueva.

La función ***showData*** ejecuta las acciones del modelo y muestra las gráficas con los datos. Para mostrar las gráficas usa la biblioteca ZingChart[2].

Antes de ejecutar la **acción** lee el tiempo que lleva de ejecución para que arranque en el momento oportuno. Para calcularlo, le pasa a una función Javascript llamada ***getRunningTime*** todos lo valores de la base de datos de esa acción y el tiempo actual, y esta función calcula el tiempo en función del último valor recibido. Si el último valor recibido es:

- START, el tiempo de ejecución es la resta del momento actual menos el *timestamp* del último valor recibido.
- STOP, no hay tiempo de ejecución.

[2] https://www.zingchart.com/

- PAUSE, vuelve a llamar a la misma función de forma recursiva pero pasando todos los valores menos el último y como tiempo el *timestamp* del último valor.
- RESUME, es una combinación de START y PAUSE, el tiempo de ejecución es la resta del momento actual menos el *timestamp* del último valor recibido, más la respuesta de volver a llamar a la misma función de forma recursiva pero pasando todos los valores menos el último y como tiempo el *timestamp* del último valor.

Aparte de visualizar la acción virtual en el modelo 3D, también se muestran distintas gráficas con los datos medidos por lo sensores o el estado de las acciones virtuales. Estas gráficas se explican en la sección 6.3.2 del Capítulo Interfaz de usuario.

5.3.2.5. Controlador de escenarios

El Controlador tiene tres funciones principales. Las dos primeras funciones coinciden con las funciones del Administrador de modelos 3D, pero con escenarios. Es decir, la primera función es crear, modificar y eliminar escenarios, y la segunda es proveer al usuario de los escenarios que pide. Por lo que también se pueden reutilizar las Figuras 5.4 y 5.5.

Al crear un escenario desde el Editor, el Controlador recibe los datos correspondientes. El primer paso a realizar en el Controlador es buscar el modelo 3D en su base de datos para leer el *path* del fichero y su extensión, y añadir estos datos al Repositorio del escenario.

El siguiente paso es añadir el paisaje al modelo. Si el usuario no ha añadido un paisaje no hace nada. Si el usuario ha añadido una única imagen como paisaje, guarda la imagen en el almacenamiento y añade al Repositorio el *path* donde se ha guardado y que es del tipo «*texture*». Si el usuario ha añadido seis imágenes –una por cada cara del cubo–, almacena todas las imágenes y añade al Repositorio un array con los seis *paths* y que es del tipo «*cube*». Por último, crea el propio escenario con su nombre, el identificador único, el propietario, el modelo con los datos modificados anteriormente y las acciones, eventos y medidas si tiene. En la Figura 5.12 se muestra un diagrama de flujo de la creación de un escenario.

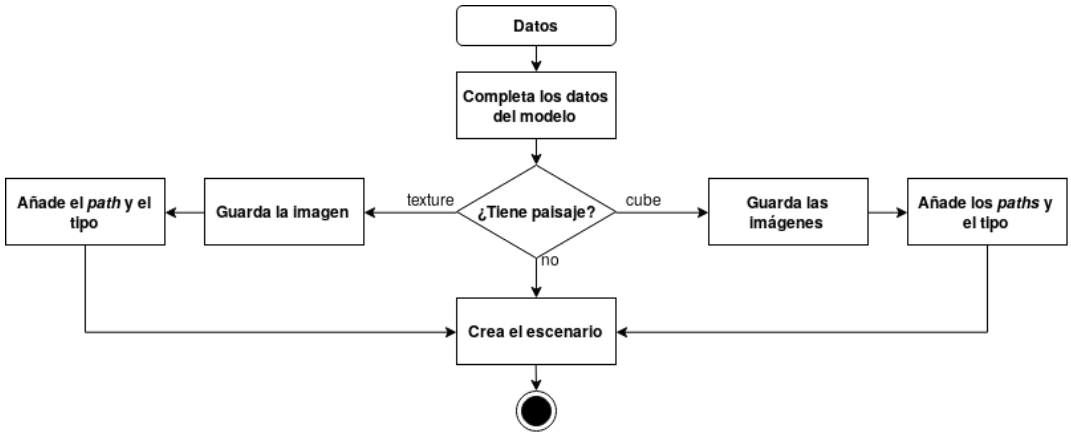


Figura 5.12: *Diagrama de flujo que explica los pasos para crear un escenario.*

La tercera función es la más importante. Se encarga gestionar los datos que se envían entre el mundo físico y la plataforma. Los datos pueden ir en dos sentidos: de la plataforma al objeto (evento) o del objeto a la plataforma (acciones y medidas). La conexión entre el Controlador y el Visualizador se realiza mediante WebSockets y la comunicación con el mundo físico por MQTT, como ejemplifica la Figura 5.13.

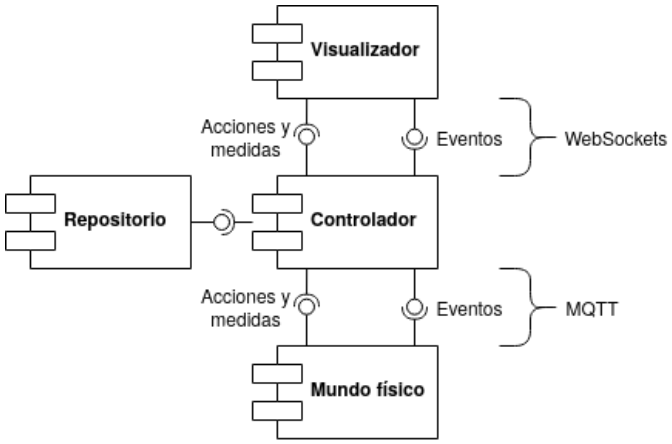


Figura 5.13: *El Controlador es un intermediario entre el objeto y su representación.*

Un evento se genera desde el Visualizador cuando el usuario interactúa con el modelo virtual. El Visualizador envía un mensaje a través del WebSockets al Controlador con datos del usuario, el escenario y el evento. El Controlador lee esos datos y construye un mensaje con el protocolo de aplicación creado para la comunicación y lo envía a través de MQTT

con el *topic* adecuado.

Las acciones y las medidas las envía el objeto físico usando el mismo método: envía un mensaje con el protocolo de aplicación creado para la comunicación a través de MQTT con el *topic* adecuado. La plataforma está siempre subscrita a todos los mensajes MQTT que le puedan llegar. Al recibir un mensaje decodifica los datos. Del *topic* extrae el usuario que lo envía, el escenario y qué datos está enviando, es decir, el nombre que tiene la acción o medida. Del *payload* del mensaje extrae los datos enviados, la longitud del mensaje y el tipo de mensaje (El tipo de mensaje hace referencia a lo explicado en la sección 5.3.2.1). Una vez extraídos los datos, los añade al escenario en el Repositorio y, si está abierto el canal WebSockets con el Visualizador, se los envía para que los muestre. La Figura 5.14 muestra dicha secuencia.

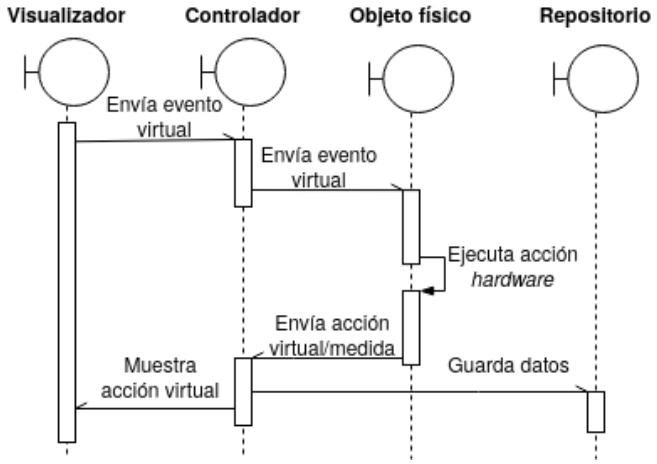


Figura 5.14: *Diagrama de secuencia con la sincronización entre objeto real y su gemelo virtual.*

Sin embargo, la gestión de los datos antes de guardarlos es diferente en función del tipo. Si los datos son de tipo «0» –es decir, una acción–, los datos solo pueden adquirir cinco valores predefinidos: START (0), PAUSE (1), RESUME (2), STOP (3) y STOP_ALL (4). Así que, el Controlador sustituye el número por su valor y lo guarda en la base datos junto a un *timestamp*. Posteriormente, envía al Visualizador el nombre de la acción y su valor. El valor STOP_ALL es especial, no está ligado a ninguna acción en concreto. Si llega este

mensaje, el Controlador para todas la acciones que se estén ejecutando y en la base de datos guarda el valor STOP en todas las acciones del escenario.

Si los datos son de los tipos del «1» al «3» –es decir, una medida con datos categóricos–, los datos solo pueden adquirir los valores de las posibles categorías, por lo que el Controlador comprueba si el valor es correcto.

Si los datos son del tipo «4» –es decir, una medida con datos numéricos discretos–, los datos solo pueden adquirir valores numéricos acotados entre un mínimo y un máximo, por lo que el Controlador comprueba que está dentro de los límites.

Si los datos son del tipo «5» –es decir, una medida con datos numéricos continuos–, los datos solo pueden adquirir valores numéricos, por lo que el Controlador comprueba que tiene el formato correcto.

Si las comprobaciones son correctas, guarda los datos en la base datos junto a un *timestamp*. Posteriormente, envía al Visualizador el nombre de la medida, el tipo y el valor junto al *timestamp* para que los muestre en las gráficas.

5.3.2.6. Repositorio de escenarios

Para los escenarios se ha creado una colección llamada «stages» que los guarda con el siguiente formato:

- **_id**: Valor del tipo *ObjectId* alfanumérico. Es único para cada dato en una base de datos. Se asigna automáticamente. Solamente se usa de manera interna.
- **actions**: Array con todas las acciones creadas en el escenario. Cada acción se compone de un nombre y un array de animaciones, donde cada animación se compone de un nombre, el número de veces que se repite y el *frame* en el que acaba.
- **data**: Array con todos los datos que han llegado, ya sea de acciones o de medidas. Cada elemento se compone del nombre de la medida o acción, el tipo de dato y un array con todos los valores que han llegado. De cada valor se guarda el valor en sí y un *timestamp* del momento en el que se guarda.

- **date_created**: Fecha de creación del escenario. Valor del tipo *ISODate*.
- **date_updated**: Fecha de la última modificación del escenario. Se inicia con la fecha de creación. Valor del tipo *ISODate*.
- **events**: Array con los datos de los posibles eventos. En cada evento se puede encontrar un nombre, el evento HTML que lo desencadena y un array con las partes del modelo donde se debe ejecutar el evento.
- **id_str**: Identificador único para cada escenario de un mismo usuario. Lo elige el usuario y se usa de forma externa. Valor del tipo *string*.
- **measurements**: Array con los datos de medidas que se pueden tomar. Cada medida consta de un nombre, su tipo (1-5) y, dependiendo del tipo, un array con los posibles estados, los valores mínimo y máximo, o las unidades.
- **model**:
  - **background**:
    - **path**: Dirección donde se guarda la imagen de *texture* o array con las seis direcciones de *cube*. El nombre se forma con un combinación del ID del usuario que ha creado el escenario y del «id_str» del escenario. Valor del tipo *string*.
    - **type**: Tipo de paisaje: «*cube*» si hay seis imágenes o «*texture*» si solo hay una. Valor del tipo *string*.
  - **ext**: Es la extensión del fichero del modelo 3D. Valor del tipo *string*.
  - **name**: Nombre del modelo 3D. Valor del tipo *string*.
  - **path**: Dirección donde se guarda el fichero del modelo 3D. Valor del tipo *string*.
  - **rotation**: Rotación del modelo 3D. Si el usuario no proporcionar ningún valor usa los mismos valores que tiene el modelo. Tiene un valor por cada eje: 'x', 'y' y 'z'. Valores del tipo numérico.

    - **scale**: Escala del modelo 3D. Si el usuario no proporcionar ningún valor usa los mismos valores que tiene el modelo. Tiene un valor por cada eje: 'x', 'y' y 'z'. Valores del tipo numérico.

- **name**: Nombre dado al escenario. Valor del tipo *string*.
- **owner_id**: Identificador del usuario que ha creado el escenario. Valor del tipo *string*.

5.4. Mundo físico

La tarea de sincronizar el objeto real con su modelo virtual depende de la programación del objeto. La plataforma se encarga de permitir al objeto ejecutar sobre ella las acciones que considere oportunas y de comunicar los eventos que sufre, que coincida una acción/evento real con una acción/evento virtual se tiene que programar en el objeto. La sincronización es una parte fundamental y es necesario prestar atención al crear la lógica del objeto.

5.4.1. Componentes necesarios en el objeto real

En el objeto real, ver Figura 5.15, se requieren elementos que intercepten los sucesos en la lógica del objeto real y realicen las acciones de comunicación necesarias.

Dado un objeto real, se asume que hay elementos asociados al procesamiento de acciones, componente **Sw actuador original**, y elementos asociados a la recepción de eventos, componente **Sw sensor original**.

Sobre estos elementos, se define un *Srapper* que intercepta los sucesos, componentes **Sensor *wrapper*** y **Sctuador *wrapper***, e integran los nuevos comportamientos necesarios para comunicarse con el **Controlador** del Administrador de escenarios. La comunicación sería con MQTT, como se indicaba también en la Figura 5.13.

La lógica dentro de los componentes **Sensor *wrapper*** y **Actuador *wrapper*** normalmente se puede reducir a quedarse a la espera de una interrupción, cuando llega valorarla y actuar consecuentemente mediante una función de *callback*.

La interrupción (o evento) puede venir del propio objeto, por ejemplo, ha detectado presencia o ha acabado el tiempo de un temporizador, o se envía desde la plataforma, es decir, ha recibido un mensaje MQTT. Como cada mensaje MQTT que le llegue al objeto real está relacionado con una interrupción real, es fácil asignar a ambos eventos el mismo *callback*. De esta manera el objeto actúa de la misma forma tanto con un evento propio como virtual.

Hay que recordar que el objeto real puede tener asociados comportamientos por defecto cuando se recibe una información por un sensor o bien algún elemento solicita actuar. Estos comportamientos hay que preservarlos.

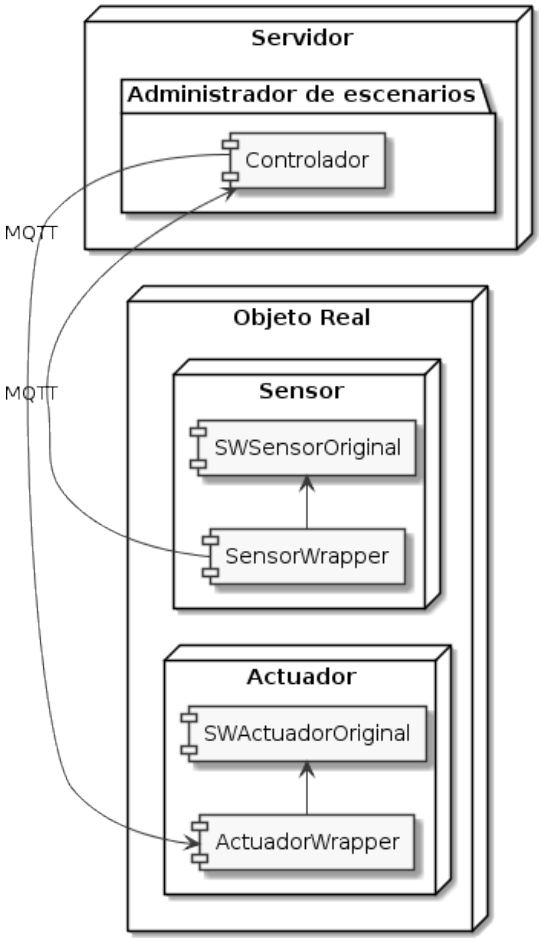


Figura 5.15: *Componentes necesarios para integrar la plataforma con objetos reales.*

Como se muestra en la Figura 5.15 hay dos módulos importantes:

- El **Sensor *wrapper*** engloba los eventos reales leídos por el objeto y los eventos virtuales que le envía la plataforma. De este modo, el origen del evento es totalmente transparente para el objeto. Tiene que escuchar los mensajes de la plataforma y las interrupciones *hardware* del dispositivo.

- El **Actuador *wrapper*** ejecuta tanto las acciones reales del objeto como las virtuales de la plataforma. Al igual que el módulo anterior, simplifica la gestión de las acciones para que no se muestren diferencias entre reales y virtuales. Maneja los actuadores físicos del dispositivo y envía las acciones virtuales a la plataforma.

De esta manera, se puede simplificar el proceso a la secuencia: llega evento ⇒ se ejecuta acción.

5.4.2. Problemática al definir la lógica del objeto

Si llegan a la vez un evento virtual y otro *hardware*, el mismo programa debe gestionar a cual se le da prioridad o cómo se quiere actuar. El encargado de elegir qué *callback* se va a ejecutar es el Sensor *wrapper*.

La función de *callback* puede realizar cualquier acción dentro de las posibles de un objeto y, si se ha creado bien, el modelo virtual tiene la capacidad de ejecutar las mismas acciones. Cada acción se ejecuta recibiendo el mensaje adecuado para cada una de ellas. Así que, para completar la sincronización, por cada acción real que ejecute el objeto, debe enviar un mensaje a la plataforma para que ejecute la misma acción. De lo que se encarga el Actuador *wrapper*. En la Figura 5.16 se muestra este comportamiento.

De esta manera, el objeto actúa de igual forma si le llega un evento real o su homónimo virtual y por cada acción física realizada ordena al modelo virtual que haga la misma. Y de esta forma se sincroniza el objeto con el modelo.

Es importante darse cuenta que cuando en una lámpara del mundo virtual se pulsa el interruptor y se ve cómo se enciende la luz, la orden no es directa. Al pulsar el interruptor se le comunica a la lámpara real que se ha pulsado el interruptor, la lámpara real enciende

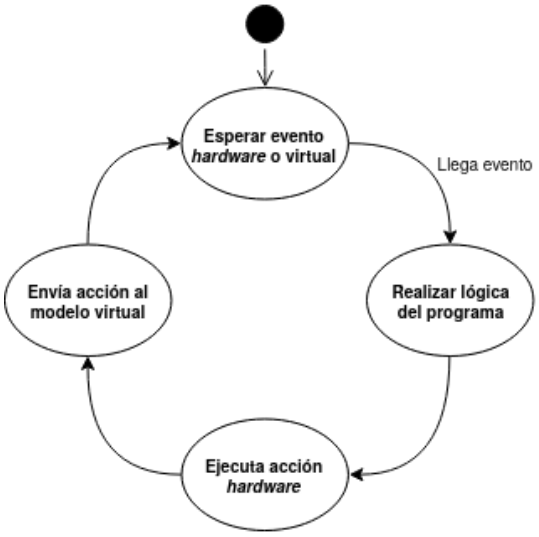


Figura 5.16: *Sincroniza de un objeto con su modelo. Al objeto real le llega un evento, ya sea hardware o virtual, y ejecuta una acción, tanto hardware como virtual.*

su luz y a la vez le ordena al modelo virtual que encienda la suya. Por eso es común que cuando se pulsa el interruptor virtual se encienda la luz real antes que la virtual y esta diferencia crece según desciende la velocidad de la comunicación. Este es un caso sencillo y puede parecer poco eficiente funcionar de esta manera, pero existen situaciones en las que es imposible que la lógica esté en la plataforma, por ejemplo, cuando la acción a ejecutar depende de factores ambientales. Usemos el caso de un aparato que al pulsar un botón mide el ruido y dependiendo de la cantidad de decibelios medidos enciende una luz verde, naranja o roja. En este caso se entiende mejor que el encargado de decidir qué luz se enciende sea el objeto real y no el virtual y que se tenga que seguir este camino.

Capítulo 6

Interfaz de usuario

6.1. Introducción

Este capítulo muestra cómo es la interfaz de los módulos que se han explicado en el Capítulo 5.

6.2. Administrador de modelos 3D

6.2.1. Visualizador de modelos 3D

El Visualizador proporciona una vista donde el usuario puede ver una lista con todos sus modelos y cada uno de ellos con más detalles.En las Figuras 6.1 y 6.2 se ven la vista de todos los modelos y de uno individual respectivamente.

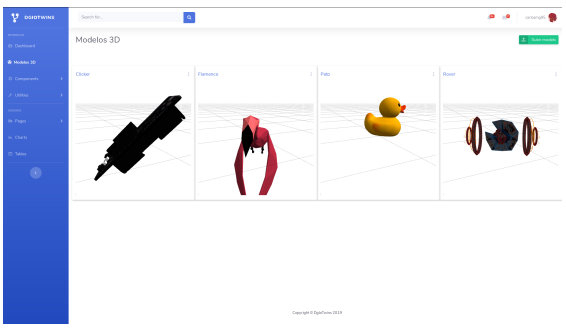


Figura 6.1: *Lista de modelos subidos por un usuario.*

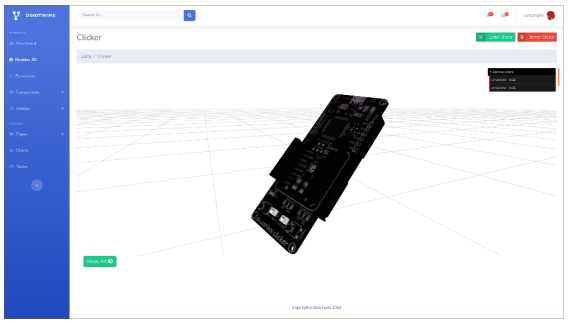


Figura 6.2: *Modelo de un 6LoWPAN Clicker desde el navegador.*

Además, la vista individual también permite ver los modelos en realidad virtual y realidad

aumentada, como se ve en las Figuras 6.3 y 6.4.

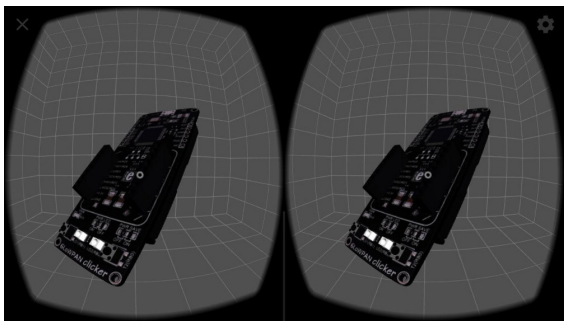


Figura 6.3: *Modelo de un 6LoWPAN Clicker en realidad virtual.*

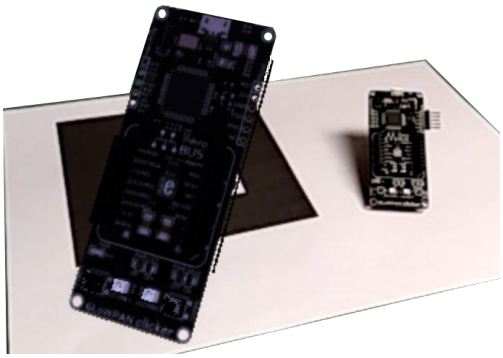


Figura 6.4: *Modelo de un 6LoWPAN Clicker en realidad aumentada junto con el real.*

6.2.2. Editor de parámetros de los modelos 3D

El Editor aporta una interfaz gráfica para subir nuevos modelos y para editar los ya existentes.

En la página de visualizar todos los modelos hay un botón de «Subir modelo» que hace aparecer el modal que se ve en la Figura 6.5 donde el usuario puede subir un fichero 3D y asignarle un nombre. En caso de que no haya nombre o esté repetido muestra un mensaje de error. También muestra un mensaje de error si no hay fichero u ocupa más de lo permitido.

En la página de visualizar un único modelo hay un botón de «Editar "NombreModelo"» que hace aparecer una campo de configuración como el que se ve en la Figura 6.6 donde el usuario puede cambiar la escala y la rotación por eje del modelo. Por cada pulsación de botón el modelo escala –en negativo o positivo– un 10 % su tamaño o gira –en el eje y dirección deseada– 0,1 radianes. Si se mantiene pulsado un botón, estos cambios se realizan cada 100 milisegundos. De ambas formas los resultados se van viendo en tiempo real según se modifican. En esta misma página hay un botón de «Borrar "NombreModelo"» que, después de pedir confirmación a través de un modal, borra el modelo.

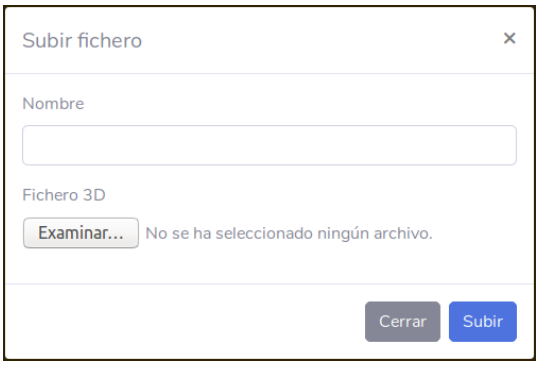


Figura 6.5: *Formularios de subida de un fichero.*

Figura 6.6: *Botones de configuración de un modelo que permiten al usuario cambiar su tamaño y rotación.*

6.3. Administrador de escenarios

6.3.1. Editor de escenarios

La función es idéntica al Editor de modelos; aporta, a través del Visualizador, una interfaz gráfica para que un usuario pueda crear sus propios escenarios y editar los ya existentes.

Los datos del escenario se organizan en distintos paneles: datos genéricos del escenario, datos del modelo asociado al escenario, datos de sus acciones, datos de sus eventos y datos de sus medidas.

1. En primer lugar, aparece el panel con los datos genéricos (Figura 6.7) con varios campos que se pueden rellenar:

   - Un **nombre** descriptivo sobre lo que se puede esperar del escenario.
   - Un **identificador** único para poder referirnos inequívocamente al escenario.
   - El **modelo** que será representado en el escenario. Si el usuario no tiene modelos subidos a la plataforma, el Editor de modelos 3D le permitirá subir uno desde la misma página. Una vez elegido el modelo aparecerá el panel con los datos del modelo (Figura 6.8).
   - Imágenes de **paisaje** –o *background*–. De manera opcional se podrá proporcionar una imagen que haga de fondo, en este caso siempre se verá la misma imagen

aunque se gire el modelo, o seis imágenes como si el modelo estuviese metido en un cubo y las imágenes fuesen cada una de las caras, en este caso al girar el modelo también gira el fondo quedando mucho más realista.

Figura 6.7: *Formulario para añadir los datos genéricos de un escenario.*

Figura 6.8: *Panel con los datos del modelo donde se puede modificar el aspecto del modelo 3D y añadir nuevos eventos, acciones y medidas.*

2. Desde el panel de los datos del modelo, el usuario puede modificar su tamaño y su rotación. Si no se cambia, se usarán los valores que tiene el modelo en ese momento. También permite añadir acciones, eventos y medidas. Los datos de las medidas hacen referencia al *Modelo de datos* de la página 54.

3. Por cada **acción** añadida aparece un panel con cabecera «Acción» como se ve en la Figura 6.9 dentro de otro panel con cabecera «Datos de acciones» donde aparecerán todas las acciones añadidas. Las acciones se pueden borrar pulsando la «X» de la esquina superior derecha. Si se borran todas las acciones, desaparece el panel contenedor.

4. En cada panel de acción hay un formulario para elegir un **nombre** que debe ser único entre todas las acciones de un mismo escenario y las **animaciones**, que por defecto aparece únicamente una pero se pueden añadir más pulsando el botón «+ Añadir animación». Si se eligen varias animaciones la acción será una combinación simultánea de todas ellas. Cada animación tiene varias opciones de configuración:

- Se elige que **animación** entre las animaciones que tiene el modelo se va a representar. Una animación no se puede repetir dentro de una misma acción.
- El número de **repeticiones** de esa animación. Si se elige cero se ejecuta indefinidamente.
- La **posición final** de la animación. Es decir, si la animación acaba en el último *frame* o vuelve al primero. Si la animación se repite indefinidamente esta opción es irrelevante.

5. Por cada **evento** añadido aparece un panel con la cabecera «Evento» dentro de otro con cabecera «Datos de eventos» donde aparecen todos los eventos y se puede ver en la Figura 6.10. Cada evento se puede borrar pulsando la «X» y si no quedan eventos desaparece también el panel contenedor.

6. Por cada evento se elige:

   - El **nombre**, que tiene que ser único entre los eventos de un escenario.
   - El **evento HTML** que desencadena el evento del escenario. Hay cuatro posibles: *click*, doble *click*, *mousein* y *mouseout*.
   - Las **partes del modelo** donde se debe ejecutar el evento. Cada modelo 3D está formado por una o varias «mallas» –o «*meshes*»–, que en el caso del 6LoWPAN Clicker que vemos en la Figura 6.10 son: botón izquierdo y derecho; base, *mikrobuses*...

   Por ejemplo, se crea un evento que se llame «pulsar_btn_izq» y se ejecute al hacer *click* en las partes que forman el botón izquierdo: «Boton_pulsador_izq» y «boton_base_izq».

7. Por último, y al igual que ocurre con acciones y eventos, cuando se añaden **medidas** aparece un panel con la cabecera «Medida» dentro de otro con cabecera «Datos de

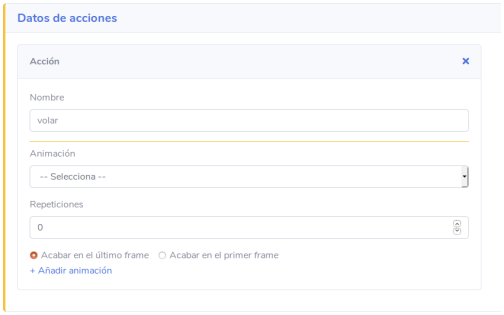


Figura 6.9: *Formulario para crear acciones dentro de un escenario.*

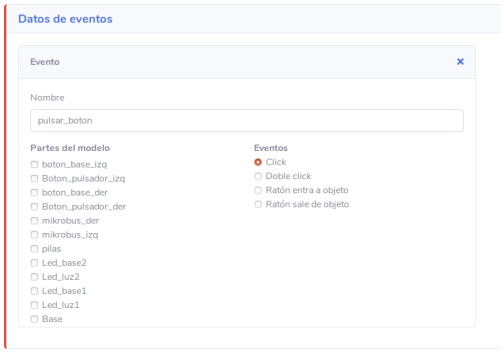


Figura 6.10: *Formulario para crear un evento dentro de un escenario.*

medidas» donde se ven todos los paneles de datos, como muestra la Figura 6.11. Se puede borrar cada dato pulsando la «X» y si ya no queda ninguno desaparece también el contenedor.

8. Lo primero que aparece es la opción de asignar un nombre –que deberá ser único– y de seleccionar el tipo de dato que quiere el usuario. En función del tipo de dato aparecerán unos campos u otros:

   - Si se elige datos **categóricos** de cualquier tipo, aparecerá un área de texto donde se deben introducir las categorías separadas por «;». En el caso de datos **dicotómicos** no permite crear más de dos. Siempre obliga a introducir al menos dos categorías.
   - Si se elige datos **numéricos discretos** aparecerán dos campos numéricos donde el usuario debe introducir los valores máximo y mínimo.
   - Si se elige datos **numéricos continuos** aparecerá un campo de texto para introducir las unidades en las que se miden de los datos.

9. Para finalizar se pulsa en el botón «Crear escenario».

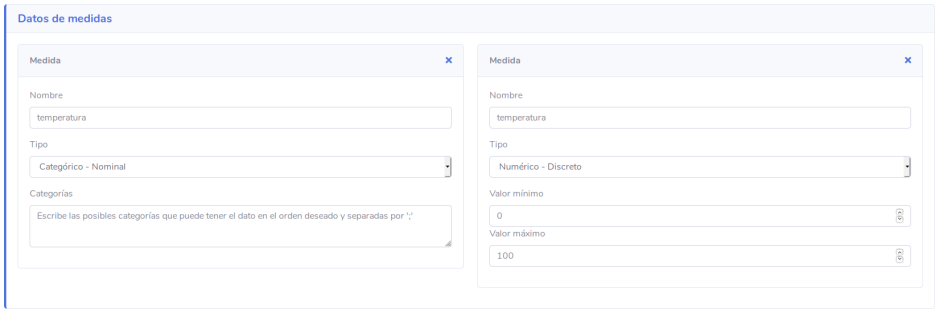


Figura 6.11: *Formularios para crear medidas de los tipos categórico nominal y numérico discreto.*

6.3.2. Visualizador de escenarios

El Visualizador de escenarios puede mostrar una lista con todos o uno individual. El Visualizador individual se compone de dos Visualizadores distintos, en uno se ve el modelo 3D y en el otro, el Visualizador de datos, se ven gráficas con datos enviados por el objeto real.

La lista de escenarios (Figura 6.12) muestra el modelo 3D de cada escenario, por lo que a simple vista puede parecer igual que el anterior, salvo que se haya elegido un paisaje.

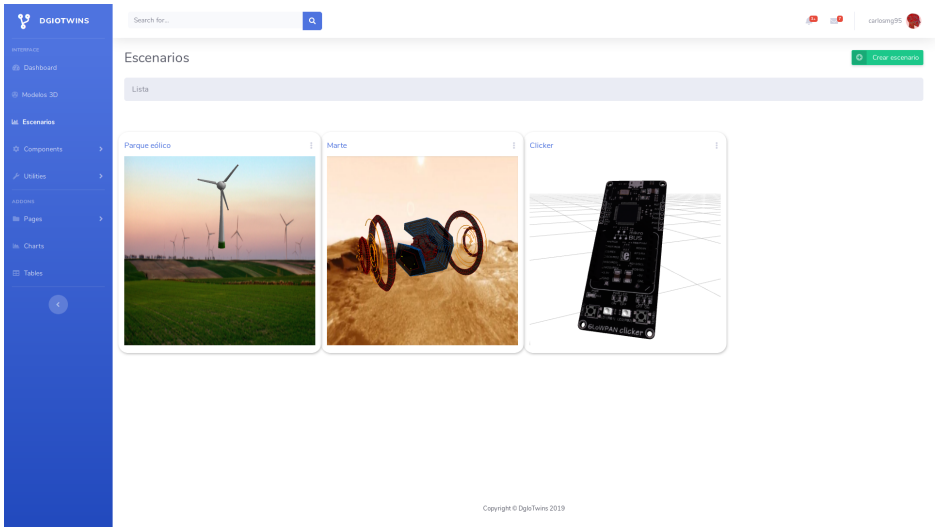


Figura 6.12: *Lista de escenarios creados por un usuario.*

Por cada acción del escenario se muestran dos gráficas. Una es una gráfica temporal que muestra los tres estados de la acción: «corriendo», «pausada» y «parada», y la otra es una

gráfica de sectores que muestra el porcentaje de tiempo que ha pasado en cada estado. En la primera gráfica se puede modificar la ventana de tiempo modificando los datos de ambas gráficas para ajustarse a la ventana. Estas gráficas se pueden ver en la Figura 6.13.

Figura 6.13: *Gráficas con los datos de la acción «volar».*

Las gráficas mostradas para cada tipo de datos de medidas son distintas:

Para las medidas con datos **categóricos** de cualquier tipo se muestra una gráfica de barras donde se pueden ver todas las categorías, en el orden elegido por el usuario, con las veces que se ha medido cada categoría, y otra de sectores donde se comparan las categorías entre sí, mostrando el porcentaje de tiempo pasado en cada una. Se puede ver un ejemplo en la Figura 6.14.

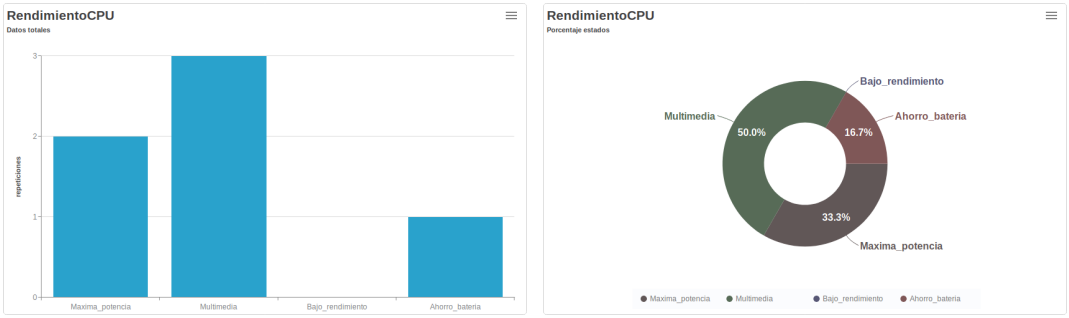


Figura 6.14: *Veces totales que una CPU ha estado en cada uno de sus modos y el porcentaje de veces que ha estado en cada modo.*

Para las medidas con datos **numéricos discretos** se muestra una gráfica de tipo aguja con el último valor recibido y otra de área con el histórico de todos los valores y varios parámetros estadísticos: media, mediana, moda y varianza; como se puede ver en la Figura 6.15.

En la segunda gráfica se puede modificar la ventana de tiempo y al hacerlo los parámetros estadísticos se ajustan a la nueva ventana.

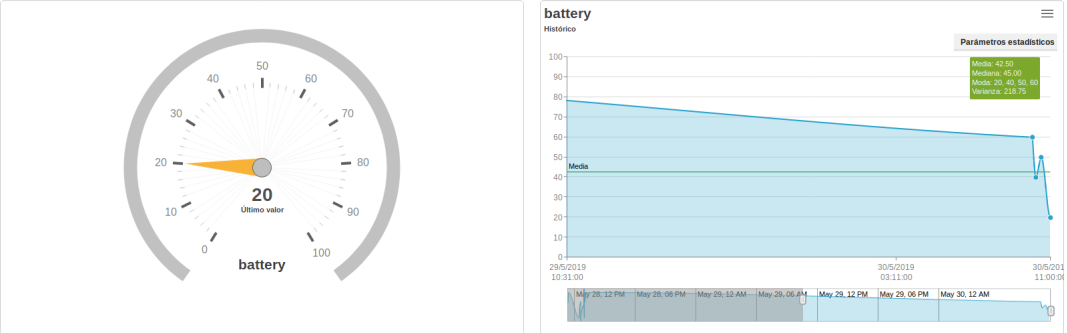


Figura 6.15: *Valor actual de batería de un dispositivo y su histórico.*

Para las medidas con datos **numéricos continuos** se muestran una gráfica lineal con los datos de forma continua en el tiempo y varios parámetros estadísticos: media, mediana, moda y varianza, y otra gráfica de barras con el acumulado de veces que se repite cada valor. Para calcular el acumulado primero se redondean los valores, por lo que si llega una temperatura de 23,4 ºC y otra de 22,9 ºC lo cuenta como si hubiesen llegado dos de 23 ºC; ya que el espacio de números reales es infinito. La ventana de tiempo de la primera gráfica se puede modificar y al modificarse cambia los parámetros estadísticos y ajusta los datos de la segunda gráfica a los datos que quedan dentro de la ventana. Se puede ver un ejemplo en la Figura 6.16.

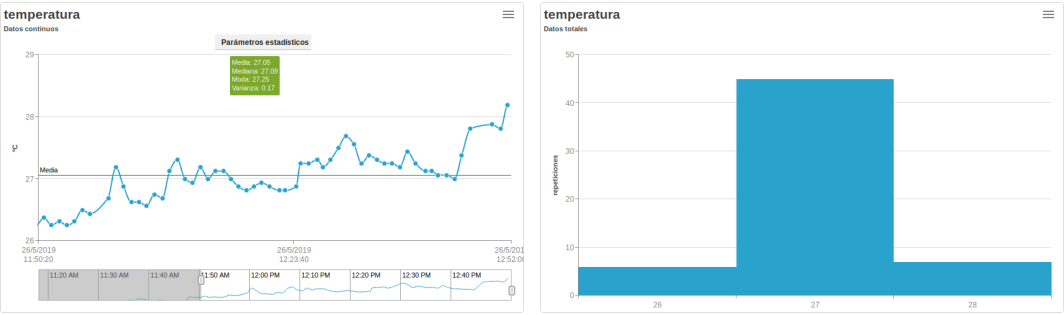


Figura 6.16: *Valores de temperatura en un cierto tiempo con varios parámetros estadísticos y las veces que se ha obtenido cada valor.*

Capítulo 7

Casos de estudio

7.1. Introducción

La mejor forma de entender el funcionamiento y las principales funcionalidades de la plataforma es mediante ejemplos prácticos. Este capítulo explica de este modo cómo funciona la plataforma.

El usuario será el actor y se explicará el proceso para empezar a usar un escenario. Para abarcar un espectro más amplio de posibilidades se realizarán dos casos de estudio.

En el primer ejemplo, se subirá el modelo de un objeto que se puede usar por sí mismo, en concreto una placa 6LoWPAN Clicker, para demostrar el potencial de la plataforma. En este ejemplo, se pulsarán los botones del dispositivo real y de su modelo virtual y se observará que los leds se encienden y se apagan cuando corresponde. El modelo virtual se visualizará en el navegador, en realidad virtual y en realidad aumentada.

En el segundo se sensorizará un objeto para poder ser usado en la plataforma y demostrar una mayor utilidad de esta. En este ejemplo, se monitorizará un aerogenerador eléctrico. Se mostrará en la plataforma cuando está encendido y cuando apagado, y la cantidad de energía que está generando. El dispositivo real se va a simular y en el modelo virtual se observarán las hélices moviéndose cuando esté encendido y una gráfica con la energía generada.

En el apéndice D se puede ver los valores de los parámetros tanto de los modelos como de los escenarios creados para estos casos de estudio.

7.2. Caso 1

Este caso de estudio tiene como objetivo demostrar el potencial de la plataforma y la manera de sincronizar los objetos con sus modelos. Se usará una placa 6LoWPAN Clicker a la que se le ha cargado un programa por el cual se enciende el led de la izquierda si se pulsa el botón de la izquierda y el led derecho si se pulsa el botón derecho. Es decir, es necesario crear dos eventos: «Pulsar botón izquierdo» y «Pulsar botón derecho», y dos acciones: «Encender led izquierdo» y «Encender led derecho». Es un programa sencillo pero que permite observar cómo se sincroniza el objeto con su modelo, ya que al pulsar los botones tanto del objeto real como del modelo virtual se encenderán tanto los leds del objeto como del modelo.

7.2.1. Pasos en la aplicación de la plataforma

En la aplicación de los principios del Capítulo 5, se contemplan los siguientes pasos:

1. Crear y subir un modelo 3D a la plataforma que represente el Clicker de la Figura 2.2. Es necesario también subir las animaciones que se vayan a utilizar, en este caso la iluminación de leds.

2. Definir el escenario, que consiste en instanciar el modelo 3D del Clicker, definir eventos y acciones relevantes para el elemento. Cada acción/elemento vendrá asociados al Clicker de la Figura 2.2.

3. Definir los componentes *software* necesarios para interceptar eventos/acciones en el Clicker según las necesidades identificadas en la sección 5.4. Debido a que el Clicker solo tiene interfaces de comunicación de corto alcance, será necesario usar un *gateway* para poder integrarse con la plataforma.

4. Experimentar. Se trata de revisar que las funcionalidades del gemelo digital en cuanto a interacción desde el mundo físico y el mundo virtual, se cumplen.

7.2.2. Paso 1. Subida del modelo

Un escenario está estrechamente unido a un modelo 3D de un objeto real, por lo que el primer paso será elegir este modelo. La interfaz de creación de escenarios le muestra al usuario los modelos disponibles, que también podría verlos en la interfaz del Visualizador de modelos. Es posible que el modelo deseado no esté disponible, por lo que el usuario deberá subirlo a través del Editor de parámetros de modelos 3D.

El usuario deberá crear el modelo con algún programa de diseño 3D como Blender (sección 2.3.3.1) y añadir las animaciones que considere oportunas. Las animaciones se basan únicamente en tres premisas: traslación, rotación y escalamiento. De esta forma se puede decir que en el *frame* 1 el objeto está en una posición con un tamaño y una rotación determinada y en el *frame* X ha cambiado alguno o todos los valores, y el programa se encarga de simular el proceso de llegar de un punto a otro.

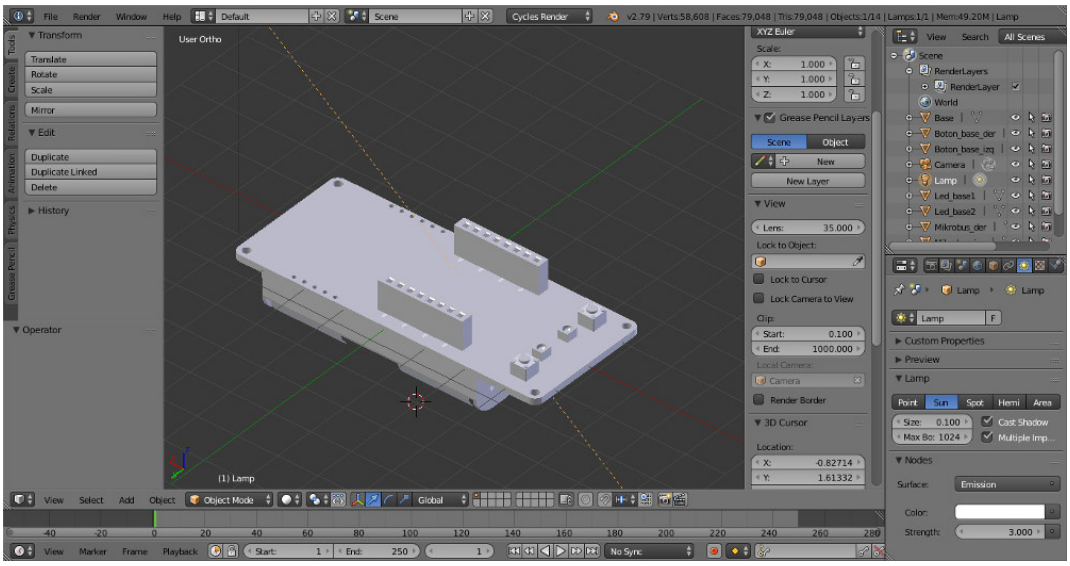


Figura 7.1: *Interfaz principal de Blender con el modelo 3D de un 6LoWPAN Clicker.*

Las animaciones de este modelo son encender el led de la izquierda (LED1) y el de la derecha (LED2). Para ello, se han creado dos esferas –una para cada led– lo suficientemente pequeñas para que se puedan ocultar en la base de cada led. Así estaría en la posición inicial. Y en la final se agrandan un poco y se suben hasta la posición deseada.

La creación de un modelo se hace a partir de un bloque principal al que se le va dando forma. En ocasiones es más fácil crear varios bloques, darles formas y luego unirlos. Por ejemplo, para este modelo se ha creado por separado la base, los *mikrobuses*, el soporte de las pilas, los leds y los botones. Al hacerlo de este modo va a permitir interactuar con cada uno de los objetos por separado, lo que interesa bastante para poder «pulsar» los botones.

Una vez subido el modelo, se puede modificar el tamaño y la rotación. Cuando esté preparado ya se puede crear el escenario.

7.2.3. Paso 2. Creación del escenario

En la interfaz del Editor de escenarios, el usuario deberá rellenar el nombre, el identificador único y elegir el 6LoWPAN Clicker como modelo. No se añade paisaje. Una vez elegido el modelo se añaden dos acciones, con una animación cada una, y dos eventos. Las acciones llevan por nombre «led1» y «led2», con las animaciones del modelo «encender_led1» y «encender_led2» respectivamente, se repiten una sola vez y acaban en el último *frame*. Para arrancar (0), pausar (1), reanudar (2) o parar (3) estas acciones, se tendrá que publicar un mensaje MQTT con el *topic* «dgiotwins/user/carlosmg95/stage/clicker/data/led1» o «.../data/led2», y con el mensaje adecuado desde el objeto real.

Los eventos tienen por nombre «click_btn_izq» y «click_btn_der», las partes del modelo son las relativas a dichos botones y el evento elegido es «*Click*». Para escuchar estos eventos el objeto real se tiene que subscribir a los *topics* «.../data/click_btn_der» y «.../data/click_btn_izq».

Una vez añadidos todos los datos se puede pulsar el botón «Crear escenario» y ya estaría creado.

7.2.4. Paso 3. Interceptando eventos en el *gateway*

En este caso no hay que sensorizar el objeto, por lo que la preparación es más sencilla. Sin embargo, como la única interfaz de red de la placa 6LoWPAN Clicker es 6LoWPAN, es incapaz de conectarse directamente al *broker* MQTT y ha sido necesario usar un *gateway*

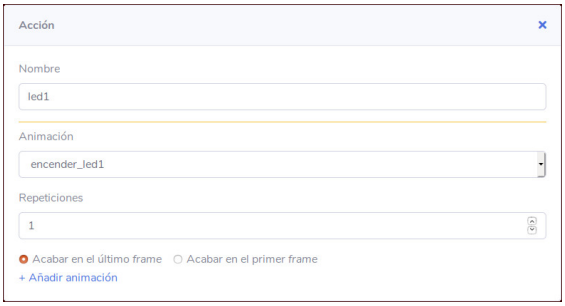


Figura 7.2: *Formulario de creación de una acción con los datos de la acción para encender el led derecho del 6LoWPAN Clicker.*

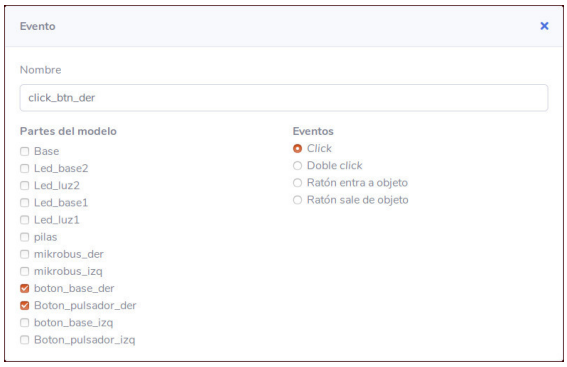


Figura 7.3: *Formulario de creación de eventos con los datos del evento «pulsar botón derecho».*

intermedio, en concreto la placa Creator Ci40. Es esta placa la que se subscribe al *topic* MQTT para luego reenviarle los mensajes al 6LoWPAN Clicker y publica los mensajes que le indica el Clicker. Para facilitar la comprensión, y dado que únicamente hace de puente, se considerará que el 6LoWPAN Clicker es el que se conecta directamente al *broker* MQTT y a la plataforma.

Para llevar a cabo el estudio se ha quemado un programa que se conecta al *broker*, se subscribe a los *topics* de los eventos y se queda a la espera de interrupciones. Estas interrupciones pueden suceder al pulsar uno de los dos botones o al recibir un mensaje. Si se pulsa el botón izquierdo o se recibe un mensaje con el *topic* relativo a pulsar el botón izquierdo, el LED1 cambia de estado y se publica un mensaje con el *topic* que activa la acción de encender el LED1 del modelo 3D, siendo el mensaje un «0» si se enciende o un «3» si se apaga. Lo mismo ocurre con el botón derecho y el LED2.

A continuación se muestra el bucle principal del código que se ejecuta en el Clicker. Se puede ver que es un bucle infinito que pone el objeto en espera hasta que llega un evento. Pueden llegar tres eventos posibles:

- Se pulsa el botón izquierdo. Entonces ejecuta el *callback* definido al principio del código: **click_btn_izq**.
- Se pulsa el botón derecho. Entonces llama a la función **click_btn_der**.

- Llega un mensaje desde el *gateway*, es decir, llega un evento virtual. Entonces lee el mensaje, ya que conoce el protocolo necesario (ver sección 3.3.1), y ejecuta el *callback* **click_btn_izq** o **click_btn_der** según corresponda.

```
while(1)
{
    PROCESS_YIELD();  // Espera hasta que llega una evento
    if (ev == sensors_event && data == &button_sensor) {
        click_btn_izq();
    }
    if (ev == sensors_event && data == &button_sensor2) {
        click_btn_der();
    }
    if (ev == tcpip_event && uip_newdata()) {
        length = udp_packet_receive(network_data, BUFFER_SIZE);

        resultado = (struct idappdata*) network_data;

        if (resultado->op == 0xd8) {
            memset(data_stage, '\0', BUFFER_SIZE);
            strcpy(data_stage, resultado->data + resultado->len);

            if (strcmp(data_stage, "click_btn_der") == 0) {
                click_btn_der();
            } else if (strcmp(data_stage, "click_btn_izq") == 0) {
                click_btn_izq();
            }
        }
    }
}
```

En el siguiente código se muestra el *callback* **click_btn_der**, que enciende el led derecho. El del led izquierdo funciona igual pero con el otro led.

Se observa que se prepara el *topic* y el mensaje que se va a enviar –el dato con el valor «3» equivale al valor STOP y el dato con el valor «0» al valor START (ver sección 3.3.1), es decir, apagar y encender– y al final lo envía. También cambia el estado de led en el propio Clicker. La diferencia con el led izquierdo es sustituir led2 por led1 en el *topic* y en la función **leds_toggle**.

```
void click_btn_der()
{
    static char topic[200];
    static struct idappdata* envio;
    envio = memb_alloc(&appdata);  // Mensaje a enviar

    envio->op = 0;  // Dato puntual

    memset(envio->data, '\0', MAXDATASIZE - ID_HEADER_LEN);
    sprintf(
        topic,
        "dgiotwins/user/carlosmg95/stage/clicker/data/led2"
    );  // Crea el topic para enviarse por MQTT


    leds_toggle(LED2);  // Cambia el estado del led derecho

    if((leds_get() & LED2) == 0) {  // Si esta encendido:
        sprintf(envio->data, "3");  // Lo apaga
    } else {  // Si no:
        sprintf(envio->data, "0");  // Lo enciende
    }

    envio->len = strlen(envio->data);
    strcat(envio->data, topic);

    udp_packet_send(
        conn,
        (char*) envio,
        ID_HEADER_LEN + strlen(envio->data)
    );
}
```

7.2.5. Experimentación con el caso de estudio

Para demostrar el potencial y la forma en la que funciona, en este caso de estudio se va a visualizar el modelo del 6LoWPAN Clicker en el navegador, en realidad virtual y en realidad aumentada. Y para hacer las comprobaciones se va a probar a pulsar los botones de la placa, del modelo del navegador y del modelo en realidad virtual para ver como el

objeto real y todas sus representaciones actúan de la misma forma.

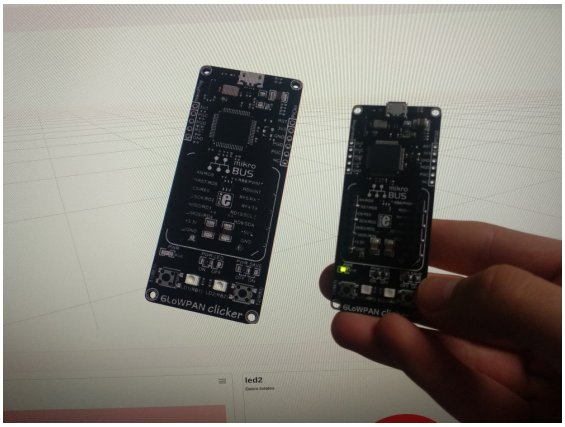


Figura 7.4: *Clicker real y su modelo en el navegador con los leds apagados.*

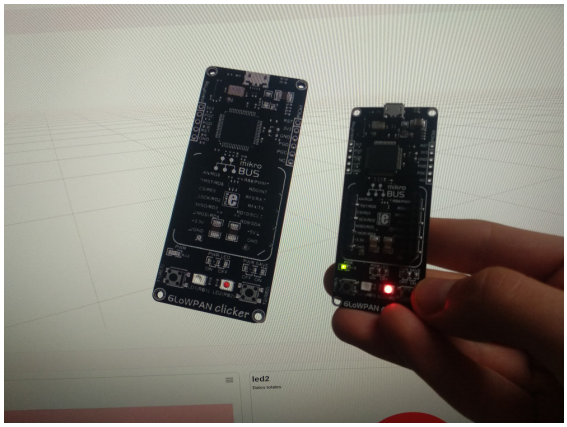


Figura 7.5: *Clicker real y su modelo en el navegador con el led derecho encendido.*

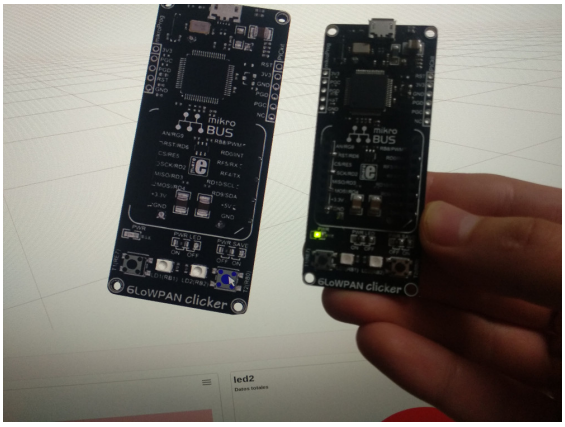


Figura 7.6: *Clicker real y su modelo en el navegador con los leds apagados y desde el navegador pulsando el botón derecho.*

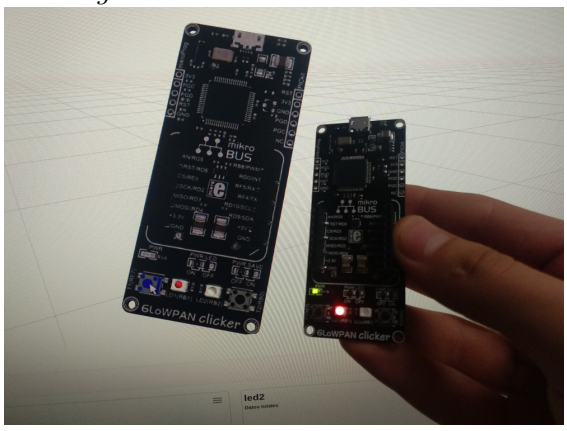


Figura 7.7: *Clicker real y su modelo en el navegador con el led izquierdo encendido y desde el navegador pulsando el botón izquierdo.*

De la Figura 7.4 a la Figura 7.7 aparecen las pruebas en el navegador. Al principio de la prueba, los dos leds están encendidos, entre la primera y la segunda imagen se ha pulsado el botón derecho del Clicker real (que es leído por el Sensor *wrapper*), lo que provoca que el Actuador *wrapper* encienda el led derecho del objeto y se envía un mensaje MQTT al Controlador de escenarios para que se encienda el del modelo. Después se pulsa el botón derecho del modelo –se puede observar en la Figura 7.6 que la flecha del ratón está sobre el botón y que el botón es de color azul, señal de que el ratón está sobre él–, lo que provoca que

el Controlador de escenarios envíe un mensaje MQTT al objeto, y se apaga el led derecho de ambos. Por último, se pulsa el botón izquierdo del modelo y se comprueba que se enciende el led de izquierdo de objeto y modelo.

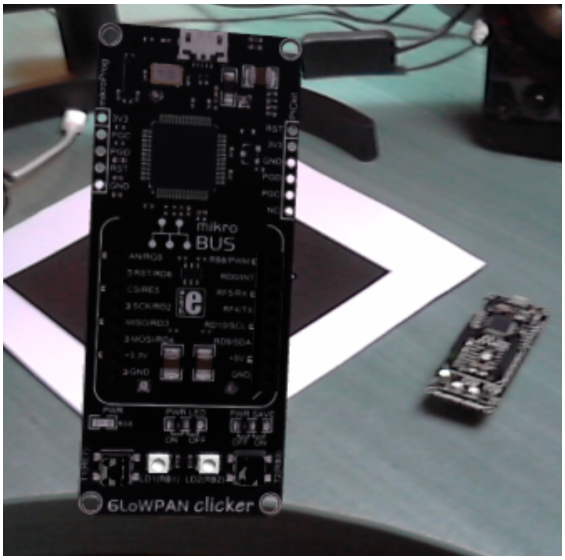


Figura 7.8: *Clicker real y su modelo en AR con los leds apagados.*

Figura 7.9: *Clicker real y su modelo en AR con los dos leds encendidos.*

La siguiente prueba se ha realizado en realidad aumentada y el resultado se puede ver en las Figuras 7.8 y 7.9. En la primera imagen aparecen los dos leds apagados, después se ha pulsado los dos botones del Clicker real (no se puede interactuar con los botones del modelo en AR) y se ve, en la segunda imagen, que ambos leds se han encendido, tanto en el objeto como en el modelo.

La última prueba se ha realizado en realidad virtual, obteniendo el resultado de las Figuras 7.10 y 7.11. Al empezar la prueba, ambos leds estaban encendidos y se ha pulsado el botón derecho del modelo en realidad virtual –se puede observar que el puntero que hay en el centro apunta al botón derecho y el botón derecho es de color azul porque está siendo apuntado–. Una vez pulsado se ha apagado el led derecho del objeto y del modelo, aunque en la imagen solo se ve el modelo.

Los resultado obtenidos han sido los esperados. Se ha podido observar que siempre hay

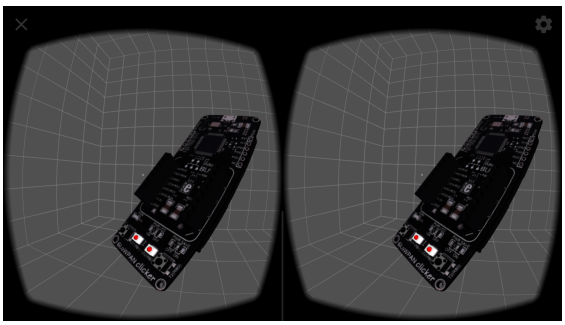


Figura 7.10: *Clicker en VR con los leds encendidos.*

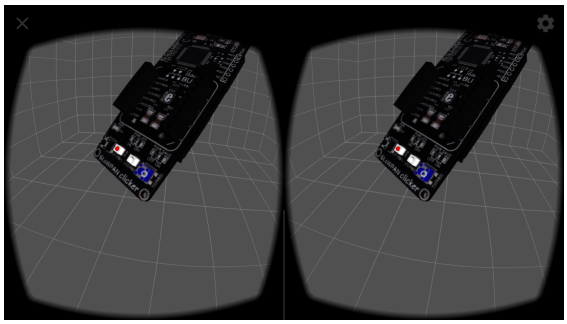


Figura 7.11: *Clicker en VR con el led derecho apagado y el puntero de realidad virtual apuntando al botón derecho.*

un poco de retraso en el modelo respecto al objeto debido a la velocidad de la red. Y si se interactúa con el modelo, los resultados se ven antes en el objeto que en el propio modelo, pero este comportamiento es el esperado debido a cómo funciona la plataforma.

7.3. Caso 2

El primer caso de estudio no muestra una situación demasiado útil. Por eso, en este caso de estudio se busca mostrar una situación real en la que un gemelo digital puede ser muy útil. Se puede dar un contexto en el que se busca gestionar micro-generación de energía renovable mediante aerogeneradores eólicos. Para controlarlo se creará un gemelo digital de un aerogenerador que mide cuándo está en funcionamiento y cuánta energía ha producido en el último periodo de tiempo. Es decir, se tiene que crear la acción «Girar» y la medida «Energía generada».

Al no poder usarse un aerogenerador real, se simulará mediante *software* el comportamiento del objeto y los datos se crearán con valores aleatorios. Pero esto demuestra que los gemelos digitales también se pueden usar para proyectar el funcionamiento final de un prototipo.

7.3.1. Pasos en la aplicación de la plataforma

Al igual que en el caso anterior, para aplicar los principios del Capítulo 5, se contemplan los siguientes pasos:

1. Crear y subir un modelo 3D a la plataforma que represente el aerogenerador. Es necesario también subir las animaciones que se vayan a utilizar, en este caso el giros de las aspas.

2. Definir el escenario, que consiste en instanciar el modelo 3D del aerogenerador, definir las acciones que puede realizar y las medidas que va a tomar.

3. Al no poder usar un aerogenerador real, se va a simular. Habrá que crear un programa *software* que tome las medidas y que realice las acciones.

4. Experimentar. Se trata de revisar que las funcionalidades del gemelo digital en cuanto a interacción desde el mundo físico y el mundo virtual, se cumplen.

7.3.2. Paso 1. Subida del modelo

Para este caso de estudio se ha descargado el molino de viento que se puede ver en la Figura 7.12 de la web Sketchfab[1] y se le han aplicado unos cambios menores con Blender. Este modelo ya tenía una animación por la cual giran las aspas.

El modelo se sube de la misma manera que el anterior.

7.3.3. Paso 2. Creación del escenario

Desde la interfaz de creación de escenarios se rellena el nombre, el identificador, se elige el molino como modelo y se añade un paisaje descargado del banco de imágenes gratuitas Pixabay[2].

[1] https://sketchfab.com
[2] https://pixabay.com/es

Figura 7.12: *Modelo 3D de un molino de viento descargado de https://sketchfab.com/3d-models/windmill-animated-6ce5667e8d5c47068ea13196036efd52.*

Figura 7.13: *Paisaje de un parque eólico descargado de https://pixabay.com/es/photos/renovables-molinos-de-viento-energía-4157213/.*

Una vez elegido el modelo, se añade una acción y una medida. El nombre de la acción es «girar», la animación es «CINEMA_4D_Main_motor_2», con repeticiones infinitas y el *frame* final es irrelevante. Para arrancar (0), pausar (1), reanudar (2) o parar (3) estas acciones, se tendrá que publicar un mensaje MQTT con el *topic* «dgiotwins/user/carlosmg95/stage/molino/data/girar» y con el mensaje adecuado desde el objeto real.

La datos medidos son los MWh de energía producida desde la última vez que se envió el dato. El nombre del dato de medida es «energia», es de tipo numérico continuo y las unidades son «MWh». Para enviar datos de esta medida se debe publicar un mensaje MQTT con el dato al *topic* «dgiotwins/user/carlosmg95/stage/molino/data/energia».

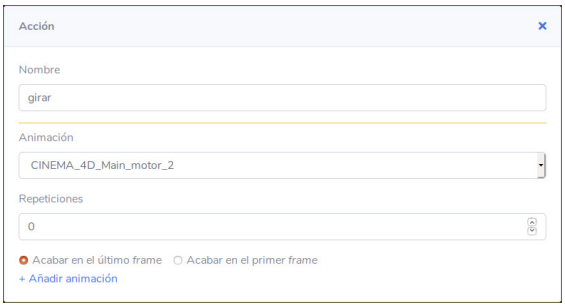


Figura 7.14: *Formulario de creación de una acción con los datos de la acción para hacer girar el motor.*

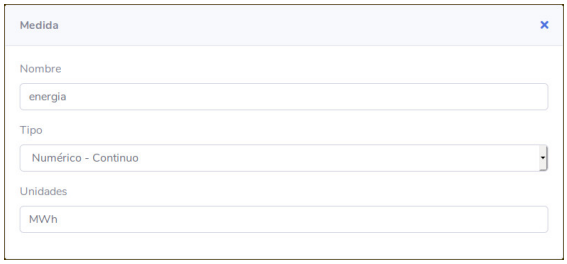


Figura 7.15: *Formulario para añadir las medidas que tomará el aerogenerador con los datos de energía.*

Una vez añadido esto, el escenario ya estaría creado. El escenario queda como la imagen de la Figura 7.16.

Figura 7.16: *Paisaje junto con el modelo 3D.*

7.3.4. Paso 3. Simulando el objeto

Para preparar este entorno habría que sensorizar un aerogenerador real, pero para el caso de estudio se va a simular con la Ci40. Así que, a continuación se explicará cómo se haría idealmente y cómo se ha hecho para probar el caso de estudio.

En un caso real, se le debe conectar al aerogenerador un sensor capaz de medir la potencia eléctrica, calcular la energía producida en un espacio de tiempo y conectarse a Internet (Al estar aislado, Sigfox es una buena idea). También es necesario conectar un sensor para saber si las aspas están en movimiento o paradas, aunque se puede reducir a que si la potencia es

cero están paradas y si es mayor están en movimiento. Cuando las aspas están en movimiento publican un mensaje MQTT para arrancar la acción correspondiente y cuando se paran un mensaje para parar la acción. Cada X tiempo, por ejemplo, cada hora, publica un mensaje con la energía producida en ese intervalo de tiempo.

En el caso de estudio hay un programa en la Ci40 corriendo a la espera de que se pulse uno de los *switch* de la placa. Al pulsarlo, las aspas cambian de estar quietas a estar moviéndose y viceversa. Mientras se estén moviendo se envían datos cada minuto con un valor de energía aleatorio.

7.3.5. Experimentación con el caso de estudio

Para demostrar la utilidad de la plataforma, en este caso de estudio se visualizará el molino a través del navegador (pudiendo sin problemas visualizarse también de cualquiera de los otros modos). Y para hacer las comprobaciones, se pulsará el *switch* de la placa para ver cómo se pone en movimiento y como cada minuto llega un nuevo valor. Para acabar, se volverá a pulsar para ver cómo se paran las aspas del modelo virtual. Es decir, cuando se pulsa el *switch* se simula que hay aire y las aspas están en movimiento, y siempre que esté en movimiento está midiendo valores de energía que se envían a la plataforma por MQTT, aunque en este caso los valores se simulan con valores aleatorios entre 2 y 13.

Para hacer pruebas en este caso de estudio, se ha encendido y apagado varias veces el aerogenerador simulado con la placa Ci40 y como resultado, además de poder ver al motor arrancar y parar, se han obtenido las gráficas de la Figura 7.17. En las gráficas se ve en qué momentos se han encendido los motores, en cuáles se han apagado, el tiempo total que ha pasado en cada estado (en movimiento o parado) y el porcentaje de tiempo de cada estado.

Además, se ha dejado durante un tiempo funcionando al «aerogenerador» para que tome algunas medidas de la energía producida. Después de unos minutos se obtienen las gráficas de la Figura 7.18. En ellas podemos ver los valores en secuencia temporal con datos de parámetros estadísticos, en concreto, la media, la mediana, la moda y la varianza, y, en la

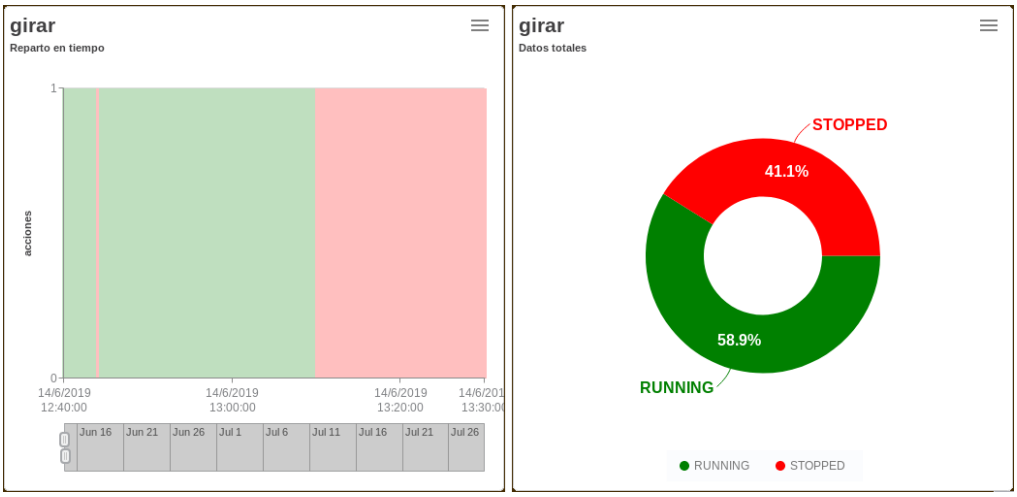


Figura 7.17: *Gráficas temporal en la que se aprecia cuando se ha iniciado o parado una acción y gráfica que muestra el porcentaje de tiempo que ha pasado en cada estado.*

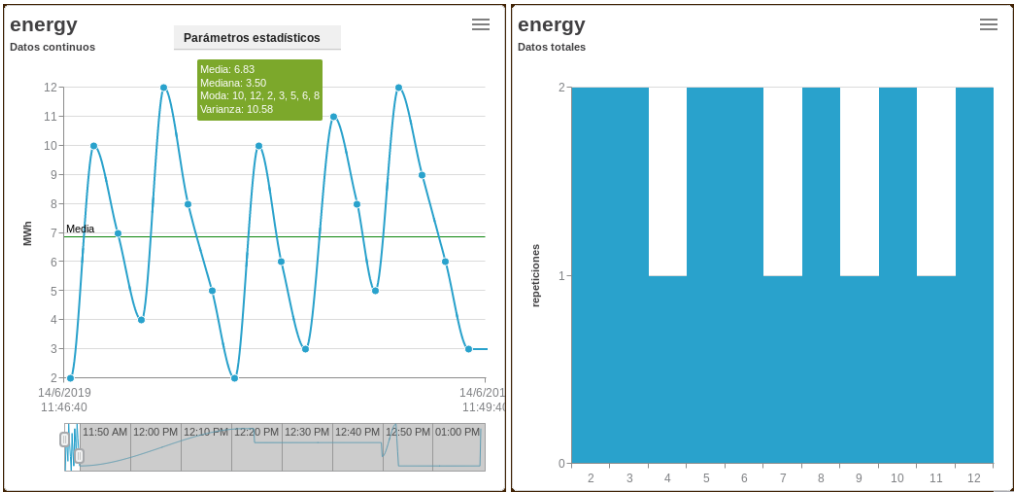


Figura 7.18: *Gráficas con la energía producida por el aerogenerador.*

otra gráfica, se ven las medidas agrupadas por valor y la cantidad de veces que se repiten. Como se explicó en el capítulo anterior, para la agrupación de los datos no se cuentan los decimales, solo números enteros, así siendo «N» un número entero, cualquier valor en el espacio $(N - 0{,}5, N + 0{,}5]$ se considerará valor «N» a la hora de mostrar las repeticiones de cada valor. Esto funciona así porque el espacio de los números reales es infinito.

Los resultados han sido satisfactorios pero hubiera sido mucho más interesante probarlos en un aerogenerador real.

Capítulo 8

Conclusiones y trabajo futuro

8.1. Introducción

Para concluir con la memoria, se resumen los conceptos que ya han sido explicados en ella. Se ha desarrollado una plataforma que virtualiza objetos reales, se comunica con ellos y es capaz de reproducir su comportamiento. Cada uno de los objetos genéricos reales tiene un modelo 3D virtual que al combinarse con los datos de un objeto en concreto forma un escenario. Haciendo un símil con la programación orientada a objetos, un modelo 3D es como la clase y los escenarios son como los objetos de esa clase.

Cada modelo 3D tiene sus propias animaciones que posteriormente serán usadas como acciones de un escenario. Cada modelo 3D está formado por distintas partes físicas con las que se podrá interactuar y generar eventos en un escenario. A cada escenario se le pueden añadir distintos datos que el objeto real puede medir, por ejemplo, litros de agua en un estanque.

Los objetos reales y la plataforma se comunican a través de MQTT. Parar que esta comunicación sea efectiva se han de usar unos *topics* concretos y un protocolo de aplicación diseñado especialmente para la plataforma.

8.2. Logros conseguidos

- **Desarrollo de un *framework* para la creación de gemelos digitales usando IoT**. Era el objetivo principal del proyecto: diseñar y desarrollar un servicio web donde los usuarios puedan subir sus modelos virtuales, vincularlos con objetos reales e interactuar con ellos.

- **Comunicación de los objetos reales y la plataforma**. Es una característica indispensable para el éxito del proyecto. La comunicación ha sido mediante MQTT.

- **Visualización de los modelos en distintos formatos**. En concreto esos formatos han sido en un navegador web, en realidad virtual y en realidad aumentada. Poder visualizar el modelo es una propiedad básica y cuanta más variedad mejor.

- **Reproducción de acciones y eventos**. Para poder llamar «gemelo» al modelo virtual es necesario que tanto el objeto real como el modelo virtual se comporten de la misma forma.

- **Recolección y visualización de datos**. Los sensores pueden enviar datos a la plataforma y esta los almacena y los muestra.

Se puede observar que se han cumplido todos los objetivos marcados al inicio del proyecto.

8.3. Problemas encontrados

- A la hora de hacer los casos de estudio se ha hecho un programa en la placa Ci40 para subscribirse y publicar mensajes MQTT. El primer intento fue hacer el programa en C pero la biblioteca Mosquitto daba problemas para subscribirse. El siguiente intento fue con Python, pero se necesita un módulo externo para usar MQTT que hay que instalar con *pip*. La placa Ci40 no tiene *pip* instalado y no se puede instalar. Así que, finalmente se ha solucionado haciendo el programa en Python, subscribiéndose a través

comandos de *shell* y para publicar llama a un programa en C diseñado para publicar mensajes MQTT con el *topic* y el protocolo de la plataforma.

- La idea inicial con el visualizador de escenarios era que el propio escenario se pudiese subscribir y publicar mensajes MQTT de forma independiente. Finalmente no fue posible y el servidor hace de intermediario comunicándose mediante WebSockets con el escenario.

- Existen algunas limitaciones como que no se puede interactuar con modelos 3D vistos en realidad aumentada, es decir, desde este modo de visualización no se pueden generar eventos.

- Hay algunos dispositivos que no tienen interfaces de comunicación que les permita comunicarse con la plataforma. Este problema se soluciona usando un *gateway* intermedio.

8.4. Trabajo futuro

Por último, se explicarán posibles características y propiedades que podrían mejorar el proyecto. Algunas no se han llevado a cabo porque estaban fuera del objetivo del proyecto y otras porque no ha dado tiempo:

- **Alertas**. Dar la posibilidad de generar una alerta cuando un valor sobrepase o esté por debajo de un límite fijado. Mostrar la alerta en la página principal y, si es muy urgente, comunicarla a través de correo, *bot* de Telegram, SMS...

- **Docker**. Meter la plataforma en un contenedor Docker para facilitar la descarga y uso de la misma.

- ***Dashboard* de administrador**. Crear una página para que el administrador del sistema pueda cambiar ciertos valores. Por ejemplo, la dirección del *broker* MQTT, el

usuario y contraseña de la base de datos, y otros ajustes de configuración. Muy útil si un usuario se descarga la plataforma.

- ***Machine learning***. Todo los datos se pueden descargar en formato CSV para que el usuario pueda aplicarles técnicas de aprendizaje automático. Pero sería interesante que se hiciesen directamente desde la plataforma.
- **Varios modelos en un escenario**. Dar la posibilidad de usar más de un modelo en un escenario. De esta manera se podría reproducir una fábrica entera con distintas máquinas en un solo escenario.
- **Datos de eventos**. Guardar también cuando se ha generado un evento para tener un registro y poder visualizarlo en una gráfica.
- **Realidad mixta**. Añadir un nuevo modo de visualizar los modelos 3D: realidad mixta.
- **Seguridad MQTT**. Hacer obligatorio el uso de usuario y contraseña para poder publicar y subscribirse a ciertos *topics*.

Chapter 9

Introduction (English)

9.1. Motivation

Nowadays, the Internet of things (IoT) is one of the most important and most potential technological areas. The improvement in the communication networks has simplified its evolution, and the "future" 5G networks are going to boost this technology because of its low latency and its speed [33].

The industry has a hight potential to take advantage of the Internet of the Things. Here appears the industrial IoT (IIoT) or Industry 4.0 [6].

Digital Twins [4, 25] are one of the most useful tools that the IoT contributes to the industry.

A digital twin is an interactive and identical virtual model of a process, service, or product. However, thanks to the IoT, a digital twin is not only a digital copy of an object. We can sensorize any object, measure all its parameters, and performance them virtually. In the same way, a user could interact with the virtual model and see the changes in the real object. The fact that the copy has all the information about the physical object enables to monitor the object and analyze the behavior. In the Figure 9.1, there are a real object and a digital object. The real object could have sensors or not. It could communicate with other devices or it could not be able to execute actions. Each situation has different challenges. The digital twin takes the information and shows it to the user. In addition, it allows the

user to interact with the 3D object and then, this modifies the real object.

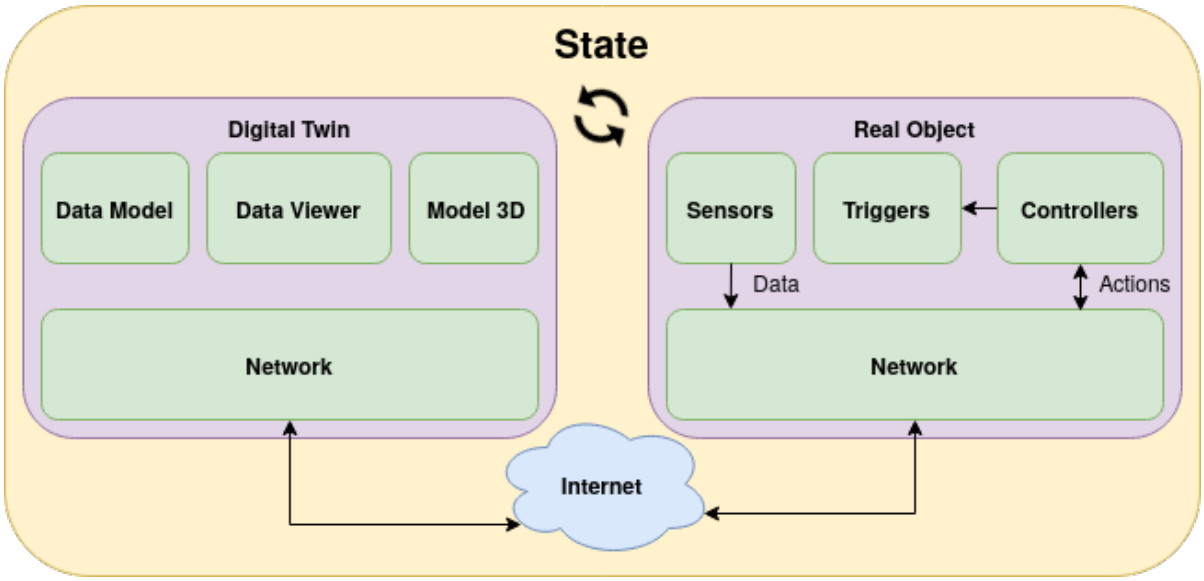


Figure 9.1: *Concept of a digital twin. The real object and its twin share the same state, and they synchronize the state through the Internet. The real object has some sensors, triggers, and controllers. The digital twin stores the data, it displays them, and it represents the 3D object.*

In general, they take advantage of the capacity of the IoT to interconnect any object to exchange data among them and with people. Thank this exchange, it is possible to monitor, control, and automatize activities. For instance, turn on a machine in a particular moment, or send an alert when the temperature of a DPC has overcome a preset threshold.

Apart from that, an user could find out problems faster and could improve the efficiency of industrial processes and machines. As a result, the industry increases its productivity, fortifies the workers' safety, improves resources management, and reduces manufacturing costs.

There are three kinds of digital twins [15, 37]: the **high-fidelity** digital twin implies that you've done an extensive analysis of the object, the **functional** digital twin indicates the basic status of an object. For example, is it on, is it off, is it full, is it empty. And the **statistical** digital twin used to collect raw data that will later be analyzed. This project is about the functional digital twins.

To uses these functional digital twins, it is necessary a list of elements that can capture data from the physical object and to communicate them to the digital twin. These elements are:

- **Smart devices**. They can be sensors, triggers, or controllers. They should be low-cost and low-energy. Their tasks are measuring data, sending data, and doing suitable actions. They are indispensable to a normal object becomes into a smart one. Figure 1.1 shows that they are really important in a real object.

- **Communications**. Networks, protocols, and communication interfaces are essential for connecting to the Internet and sharing device data with other devices and with the central server. The real object and its twins need communication to synchronize their status.

- **Data infrastructure**. Digital twins need an infrastructure that concentrates sensors data and saves them. This goal would be impossible without this infrastructure. It could be a public cloud service or a private cloud service.

- **User interface**. Besides communication between devices, a human user should see the data and interact with them. A user interface is very important. The best way to represents so many data is through charts, alerts, and statistical parameters. Another way is the 3D representation and digital twins. There are some methods to visualize digital twins: display center, virtual reality, augmented reality, and mixed reality. We have to choose a style based on needs and technology [38].

- Sometimes, there is a **data analyzer**. The infrastructure needs an app that analyzes raw data and turns them into relevant information. This element worths a digital twin, but it is not vital for developing the digital twin's task. The combination of digital twins, big data, and machine learning could be very interesting to process real-time data and makes decisions.

By combining these elements, it should be possible to synchronize the real object and the virtual model so that both perform the same actions. However, there are more urgent challenges, like rendering virtual 3D models, interacting with them, or communicating the platform with the object.

9.2. Project Goals

The main aim of the project is to develop a framework to create and to visualize digital twins using IoT. The 3D models will be visualized in the browser thank three.js [19], in virtual reality thank WebVR[1], and in augmented reality thank AR.js [7].

The principal goals of the project are:

- **Visualizing** 3D models of real objects. The platform should show them on different ways: the browser, **virtual reality**, and **augmented reality**. Besides, a user has to **interact** with them.
- **Synchronizing** the object with the model to perform the same manner. So, they must **communicate** among them.
- **Managing** the effect of the actions on the twin. The platform must allow the user to set up the animations that will be displayed for each action.

To prove that it works, a use case is proposed that will measure simple sensors and execute activators such as engines and LEDs. This project doesn't propose specific ways of programming the real objects. However, it shows some examples and the necessary architecture.

The code of the platform and the use cases is online in GitHub. The platform is in the branch "dev" of "https://github.com/carlosmg95/TFM_Digital_Twins/tree/dev" with MIT license. The use cases are in "https://github.com/carlosmg95/TFM_Digital_Twins_Casos" without license.

[1] https://webvr.info/

9.3. Structure of this Document

This section is a very brief summary of all chapters of this document. The structure is:

Chapter 1 is the project introduction. It describes the motivation, the goals, and the structure of the essay.

Chapter 2 shows the state of the art and the enabling technologies. It will analyze these technologies to create a base about them and to create a context of the main idea.

Chapter 3 explains the more important semantic of the project. Mainly, it explains a real object, a virtual model, and their communication to syncronize.

Chapter 4 shows the requirements that are necessary to develop the project and the chosen solutions.

Chapter 5 shows the architecture of the platform in detail.

Chapter 6 shows the user's interface of the developed platform.

Chapter 7 develops two use cases from the configuration to the execution. It explains the global concept.

Chapter 8 summarizes the conclusions, the problems found, and the future work.

Chapter 9 is the Chapter 1 in english.

Chapter 10 is the Chapter 8 in english.

Chapter 10

Conclusions and future work (English)

10.1. Introduction

To conclude the essay, it summarizes the concepts that have already been explained. The developed platform can virtualize real objects, communicate with them, and imitate their behavior. Each real generic object has a 3D virtual model. This model can combine with the data of a particular object, and they create a stage. For example, if we compare with object-oriented programming, a 3D model is like a class, and a stage is like an object of a class.

Each 3D model has its animations that will then be used as actions in the stages. Each model consists of different physical parts with which a user will be able to interact and generate events. Each stage can receive data measured by the real object. For instance, the amount of water in a tank.

The real objects communicate with the platform through MQTT. There are two main elements to have efficient communication: specific MQTT topics and an application protocol designed for the platform.

10.2. Achievements

- **Development of a framework to create digital twins with IoT**. This was the main goal of the project: designing and developing a web service where users can

upload their virtual models, link them with real objects, and interact whit them.

- **Communication between the real objects and the platform**. It is a really important feature to achieve the objective of the project. They communicate through MQTT.
- **Display of the models in different ways**. Specifically, these ways are: the browser, virtual reality, and augmented reality. Showing the models is a basic feature, and the more ways, the better.
- **Execution of actions and events**. If it is called "twin", the real object and the virtual model should work the same way.
- **Capture and display of data**. The sensors of the objects can send data to the platform. The platform can store and display them.

Every objective set at the beginning of the project has been achieved.

10.3. Problems faced

- In the use cases, the Creator Ci40 development board had a program that can subscribe and publish MQTT messages. Firstly, it used a C library called Mosquitto, but problems to subscribe appeared. Secondly, it tried to use Python. Python needed to install with *pip* an external module to use MQTT. The board didn't have installed *pip* and it couldn't be installed. So, in the end, the solution consists of a Python program that uses shell commands to subscribe and uses a C program to publish MQTT messages. The C program is designed to uses the correct topic and the protocol of the platform.
- Subscribing and publishing MQTT messages independently from the stages display was the first idea. Finally, this wasn't possible and the server is a middleware that communicates with the stage through WebSockets.

- There are some limitations. For example, a user can't interact with a 3D model when it is displayed in augmented reality. In this way, the model can't generate events.
- Some devices don't have communication interfaces that allow them to communicate with the platform. A middle gateway can solve this problem.

10.4. Future work

In the end, this section explains some possible features and properties that could improve the project. Some of them were out of the goals of the project, and others haven't been done for time:

- **Alerts**. Generation of an alert when a measured value is greater or less than an appointed value. The alert should be displayed on the main page or sent through email, Telegram bot, SMS...
- **Docker**. Use Docker to ease the download and the use of the platform.
- **Admin dashboard**. Create a page where the system admin can change some values. For instance, the broker MQTT direction, the user and the password of the database, and other configurations. If a user downloaded the platform, this would be very useful.
- **Machine learning**. All data can be download in a CSV file so that the user can apply machine learning techniques. However, this could be done directly from the platform.
- **Several models in a stage**. Add several 3D models in a single stage. This way, a user could create a complete factory with only one stage.
- **Event data**. Store generated events to keep a register and can display in a chart.
- **Mixed reality**. Add a new way to display the 3D models: mixed reality.
- **MQTT security**. Use user and password to subscribe and publish on some topics.

Bibliografía

[1] Ronald T Azuma. A survey of augmented reality. *Presence: Teleoperators & Virtual Environments*, 6(4):355–385, 1997.

[2] Microsoft Azure. Introducción a Azure Digital Twins. https://docs.microsoft.com/es-es/azure/digital-twins/about-digital-twins Última visita: 05/11/2019, 2019.

[3] Blender. https://www.blender.org/ Última visita: 9/06/2019.

[4] Arquimedes Canedo. Industrial IoT lifecycle via digital twins. In *Proceedings of the Eleventh IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis*, page 29. ACM, 2016.

[5] Oracle Corporation. Developing Applications with Oracle Internet of Things Cloud Service. https://docs.oracle.com/en/cloud/paas/iot-cloud/iotgs/learn-oracle-iot-digital-twin.html Última visita: 26/07/2019, 2019.

[6] José Luis Del Val Román. Industria 4.0: la transformación digital de la industria. In *Valencia: Conferencia de Directores y Decanos de Ingeniería Informática, Informes CODDII*, 2016.

[7] Jerome Etienne. *AR.js*. https://github.com/jeromeetienne/AR.js/blob/master/README.md Última visita: 9/06/2019.

[8] Eclipse Foundation. Eclipse Ditto. https://www.eclipse.org/ditto/intro-overview.html Última visita: 26/07/2019, 2019.

[9] Daniel García García. Estudio de 6LoWPAN para su aplicación a Internet de las Cosas. Master's thesis, Universidad de La Laguna, 2015.

[10] Gabriel García. Soluciones de Microsoft para IoT (webinar). Technical report, Efor, noviembre 2018.

[11] Google. https://vr.google.com/cardboard/ Última visita: 7/08/2019.

[12] Khronos Group. glTF-Blender-IO. https://github.com/KhronosGroup/glTF-Blender-IO Última visita: 9/06/2019.

[13] Khronos Group. WebGL. https://www.khronos.org/webgl/wiki/Main_Page Última visita: 9/06/2019.

[14] José Ignacio Gómez. 6LowPAN (Apuntes de la asignatura RPI1). Technical report, Facultad de Informática. Universidad Complutense de Madrid, 2018.

[15] Brian Holak. Digital twins: The next step on your IoT journey. Technical report, TechTarget, 2017. https://searchcio.techtarget.com/feature/Digital-twins-The-next-step-on-your-IoT-journey Última visita: 14/04/2019.

[16] HTC. https://www.vive.com/eu/ Última visita: 7/08/2019.

[17] Marco Hutter and Khronos Group. *glTF 2.0 Quick Reference*. Khronos Group, Febrero 2019.

[18] Javier Penalva. Qué gafas de realidad virtual (VR) comprar: guía de compras con todas las opciones según tu equipo y presupuesto. *Xakata*, 2017. https://www.xataka.com/realidad-virtual-aumentada/que-gafas-de-realidad-virtual-vr-comprar-guia-de-compras-con-todas-las-opciones- Última visita: 7/08/2019.

[19] Three JS. *three.js*. https://threejs.org/docs/index.html Última visita: 9/06/2019.

[20] Imagination Technologies Limited. *Ci40 Hardware User Guide v2*, Abril 2016.

[21] Diego Marcos, Kevin Ngo, and Don McCurdy. A-Frame. https://aframe.io/ Última visita: 9/06/2020.

[22] Microchip. *PIC32MX330/350/370/430/450/470*.

[23] MikroElektronika. *6LowPAN clicker a new idea just a click away*.

[24] Andreas F Molisch, Kannan Balakrishnan, Chia-Chin Chong, Shahriar Emami, Andrew Fort, Johan Karedal, Juergen Kunisch, Hans Schantz, Ulrich Schuster, and Kai Siwiak. Ieee 802.15. 4a channel model-final report. *IEEE P802*, 15(04):0662, 2004.

[25] Lidia Montes. Gemelos digitales en la fábrica 4.0. *El Mundo*, 2016. https://www.elmundo.es/economia/2016/01/05/568bb9b0ca4741ba2e8b457a.html Última visita: 14/04/2019.

[26] Rubén Darío Morelli, Hernán Alfredo Pangia Ctenas, and Luis Sebastián Nieva. Modelado paramétrico 3d, render y animación con software libre: Interacción freecad+ blender. *Geometrias & Graphica 2015 Proceedings*, pages 023–036, 2015.

[27] Grupo OASIS. MQTT. https://mqtt.org/ Última visita: 1/12/2019.

[28] Oculus. https://www.oculus.com/rift-s/ Última visita: 7/08/2019.

[29] Susan Opt and Christy-Dale L Sims. Scrum: Enhancing student team organization and collaboration. *Communication Teacher*, 29(1):55–62, 2015.

[30] Lidia Orellana. ¿Cómo podemos analizar los datos de nuestra empresa? Primeros pasos. Technical report, La Innovación Necesaria, 2017. https://www.lainnovacionnecesaria.com/como-podemos-analizar-los-datos-de-nuestra-empresa/ Última visita: 23/05/2019.

[31] Scott D Roth. Ray casting for modeling solids. *Computer graphics and image processing*, 18(2):109–144, 1982.

[32] Samsung. https://www.samsung.com/es/wearables/gear-vr-sm-r325nzvaphe/ Última visita: 7/08/2019.

[33] Philipp Schulz, Maximilian Matthe, Henrik Klessig, Meryem Simsek, Gerhard Fettweis, Junaid Ansari, Shehzad Ali Ashraf, Bjoern Almeroth, Jens Voigt, Ines Riedel, et al. Latency critical iot applications in 5g: Perspective on the design of radio interface and network architecture. *IEEE Communications Magazine*, 55(2):70–78, 2017.

[34] Sony. https://www.playstation.com/es-es/explore/playstation-vr/ Última visita: 7/08/2019.

[35] Jonathan Steuer. Defining virtual reality: Dimensions determining telepresence. *Journal of communication*, 42(4):73–93, 1992.

[36] Kei Studios. A quick guide to Degrees of Freedom in Virtual Reality. https://kei-studios.com/quick-guide-degrees-of-freedom-virtual-reality-vr/ Última visita: 7/08/2019, 2018.

[37] Alfonso Velosa. Digital Twins Will Impact Economic and Business Models. *Gartner*, 2017.

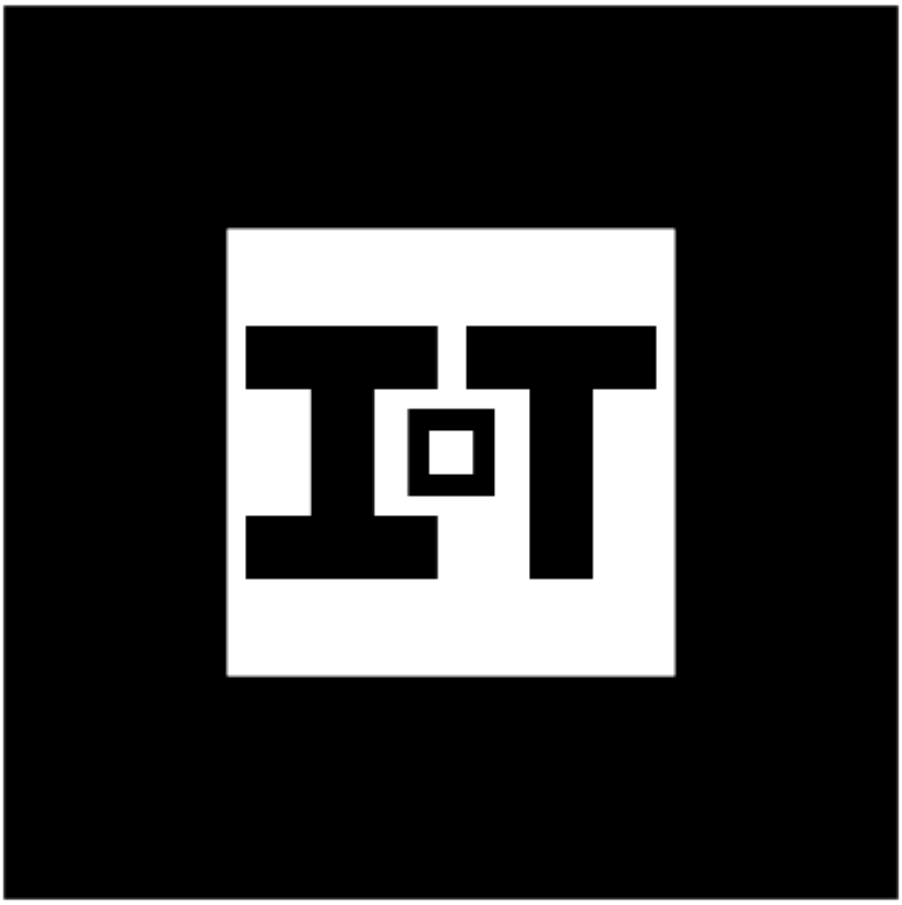
[38] VisualTechLAB. ¿Qué es un gemelo digital? Industria 4.0. Technical report, Visual Technology LAB, 2017. https://www.vt-lab.com/que-es-un-gemelo-digital/ Última visita: 14/04/2019.

[39] WebVR. https://webvr.info/ Última visita: 9/06/2019.

[40] Wikipedia. Webvr. https://es.wikipedia.org/wiki/WebVR Última visita: 9/06/2019.

Apéndice A

Patrón para realidad aumentada



Apéndice B

Módulos NPM

Como proyecto realizado en Node, para gestionar paquetes usa el sistema npm[1]. En este apéndice se van a enumerar brevemente lo módulos usados:

- **ar.js**. Módulo para crear objetos en realidad aumentada. Licencia MIT.
- **async**. Módulo para gestionar funciones asíncronas. Licencia MIT.
- **body-parser**. Módulo para parsear datos de peticiones. Licencia MIT.
- **bootstrap**. Biblioteca para el diseño de páginas web. Licencia MIT.
- **colors**. Módulo para dar color a la salida por consola. Licencia MIT.
- **cookie-parser**. Módulo para gestionar y parsear las *cookies*. Licencia MIT.
- **crypto-js**. Módulo para encriptar. Licencia MIT.
- **debug**. Módulo para ayudar a la depuración. Licencia MIT.
- **ejs**. Módulo para añadir plantillas embebidas con JavaScript. Licencia Apache-2.0.
- **express**. Módulo para crear proyecto con Node fácilmente. Licencia MIT.
- **express-fileupload**. Módulo para subir ficheros desde un formulario. Licencia MIT.

[1] https://www.npmjs.com/

- **express-partials**. Módulo para separar en partes los ficheros de la web. Sin licencia.
- **express-session**. Módulo para crear sesiones de usuario. Licencia MIT.
- **fs**. Módulo para gestionar el sistema de ficheros. Licencia ISC.
- **glob**. Módulo para buscar ficheros en función de patrones usados por la *shell*. Licencia ISC.
- **jquery**. Biblioteca para interactuar con ficheros HTML de forma sencilla. Licencia MIT.
- **method-override**. Módulo para poder usar los métodos PUT y DELETE. Licencia MIT.
- **mongodb**. Módulo para usar bases de datos MongoDB. Licencia Apache-2.0.
- **morgan**. *Middleware* HTTP para peticiones de *log*. Licencia MIT.
- **mqttr**. Módulo para enrutar mensajes MQTT. Licencia MIT.
- **path**. Módulo para trabajar con el *path* de ficheros y directorios. Licencia MIT.
- **serve-favicon**. Módulo para mostrar un *favicon*. Licencia MIT.
- **socket.io**. Módulo para trabajar con WebSockets en tiempo real. Licencia MIT.
- **startbootstrap-one-page-wonder**. Plantilla de Bootstrap usada en la página principal. Licencia MIT.
- **startbootstrap-sb-admin-2**. Plantilla de Bootstrap usada en las páginas de usuarios. Licencia MIT.
- **three**. Módulo para gestionar y visualizar modelos 3D. Licencia MIT.
- **underscore**. Biblioteca para ayudar con algunas funciones comunes. Licencia MIT.

- **zingchart**. Módulo para construir gráficos. Sin licencia.

Apéndice C

Protocolos e interfaces de comunicación

C.1. Introducción

En este apartado se explicarán los protocolos e interfaces de comunicación que se han usado en este proyecto.

C.2. MQTT

MQTT [10, 27] es un protocolo binario de comunicación *machine to machine* (M2M) muy usado en IoT y basado en TCP/IP. Es muy ligero, fácil de implementar y es idóneo para el uso en aplicaciones que necesitan muy poco ancho de banda, por eso se usa tanto en IoT. El tamaño máximo de mensaje es de 256 bytes.

Su arquitectura es de forma de estrella y sigue un modelo publicación/subscripción, desacoplando así a todos los nodos de comunicación. El centro de la estrella es un ***broker*** que se encarga de gestionar la red, transmitir los mensajes y mantener activo el canal. Todos los nodos publican sus mensajes sobre el *broker* y el *broker* los reenvía a los subscriptores. Se hace de forma asíncrona, es decir, los clientes no se bloquean mientras esperan un mensaje.

Todos los mensajes publicados incluyen un ***topic*** al que se pueden subscribir los nodos. Todos los dispositivos subscritos recibirán el mensaje. Los *topics* se estructuran de forma jerárquica, un ejemplo de *topic* podría ser: «micasa/salon/temperatura». Es posible subscribirse a varios *topics* usando *wildcards*: «#» se pone al final y «+» entre medias; así, si

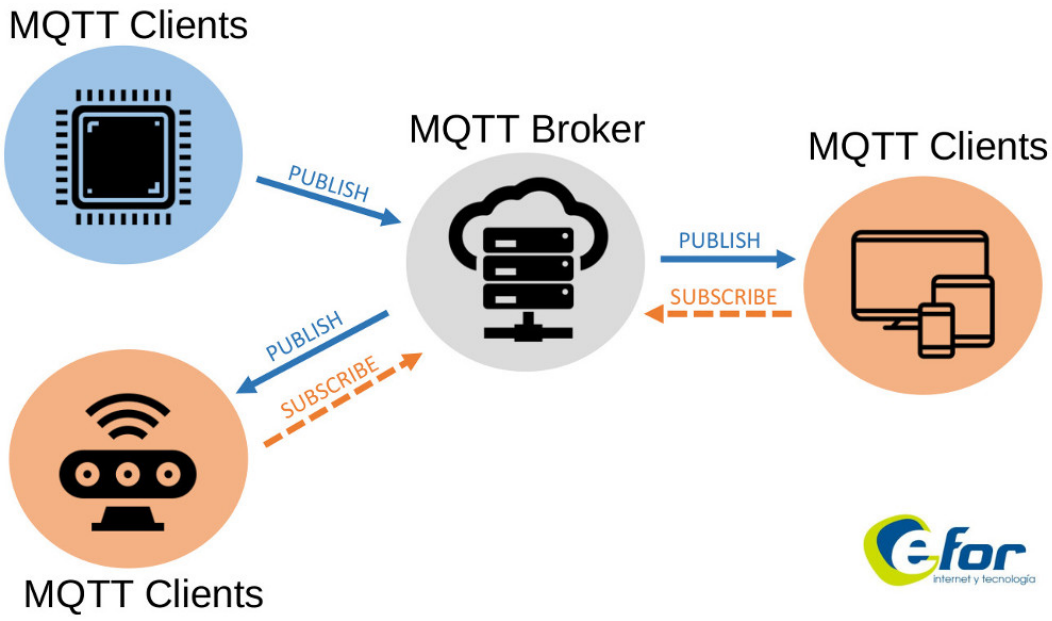


Figura C.1: *Arquitectura MQTT [10].*

un usuario se subscribe a «micasa/#» recibirá mensajes de todos los sensores de todas las habitaciones de su casa y si se subscribe a «micasa/+/temperatura» recibirá las medidas de los sensores de temperatura de todas las habitaciones de su casa.

Existen tres niveles de calidad de servicio (**QoS**):

1. *At most one* (0): No hay garantía de entrega.
2. *At least one* (1): Se garantiza que el mensaje va a llegar al menos una vez, pero es posible que llegue repetido.
3. *Exactly one* (2): El mensaje llega únicamente una vez a cada receptor.

MQTT cuenta con seguridad a nivel de red, a nivel de transporte (TLS/SSL) y a nivel de aplicación mediante el uso de credenciales.

En este proyecto se ha usado **Eclipse Mosquitto**[1]. Mosquitto es un *broker* de código abierto para implementar MQTT. Aparte del *broker*, también proporciona una biblioteca en C y órdenes de líneas de comando para el uso de clientes.

[1] https://mosquitto.org/

C.3. WebSockets

El protocolo HTTP en el que se basa la mayor parte de Internet es un protocolo petición/respuesta en el que el cliente le tiene que hacer una petición al servidor para poder obtener algún servicio, de ningún modo el servidor se puede comunicar directamente con el cliente. WebSockets[2] es un protocolo de comunicación sobre TCP que permite una conexión bidireccional *full-duplex* entre cliente y servidor. Esto permite que el servidor pueda enviar información al cliente a través de un *socket* sin la necesidad de que el cliente haga ninguna petición.

C.4. 6LoWPAN

6LowPAN [9, 14] (IPv6 *Low-Power Wireless Personal Area Networks*) es una tecnología de comunicación situado entre las capas de transporte y red que busca reducir al máximo el consumo de energía. Trabaja sobre IPv6, usa estándares abiertos, tiene integración transparente a Internet y una escalabilidad global.

Es capaz de autoconfigurarse con protocolos de arranque y descubrimiento, es decir, no es necesario configurar la red; una vez encendido, el dispositivo buscará cuáles son sus vecinos y cuál tiene que ser el próximo salto para llegar al «nodo padre».

Como protocolo de enlace usa el IEEE 802.15.4 [24], que usa las bandas de 2,4 GHz y 900 MHz, tiene una velocidad máxima de 250 kbps y su rango es de corto alcance, entre otras características.

[2] https://www.websocket.org/aboutwebsocket.html

Apéndice D

Definición de modelos y escenarios

En este apéndice se definirán los datos de los modelos y escenarios usados para los casos de estudio.

D.1. Definición de modelos

- 6LoWPAN Clicker
  - **Nombre**: Clicker
  - **Fichero 3D**: clicker.glb
  - **Escala**: 1
  - **Rotación**:
    - x: 0
    - y: 0
    - z: 0
- Aerogenerador
  - **Nombre**: Molino
  - **Fichero 3D**: windmill.glb
  - **Escala**: 0.001

- **Rotación**:
  - x: 0.1
  - y: 1
  - z: 0

D.2. Definición de escenarios

- Escenario de 6LoWPAN Clicker
  - **Nombre**: 6LoWPAN Clicker
  - **Identificador**: clicker
  - **Modelo**: Clicker
    - **Escala**: 1
    - **Rotación**:
      - x: 0
      - y: 0
      - z: 0
  - **Paisaje**:
  - **Acciones**:
    - Nombre: led1
      - Animación: encender_led1
      - Repeticiones: 1
      - *Frame* final: último
    - Nombre: led2
      - Animación: encender_led2
      - Repeticiones: 1

        - ◇ *Frame* final: último
    - **Eventos**:
        - Nombre: click_btn_izq
            - ◇ Partes del modelo: boton_base_izq, Boton_pulsador_izq
            - ◇ Evento HTML: *Click*
        - Nombre: click_btn_der
            - ◇ Partes del modelo: boton_base_der, Boton_pulsador_der
            - ◇ Evento HTML: *Click*
    - **Medidas**:

- Escenario de Aerogenerador
    - **Nombre**: Parque eólico
    - **Identificador**: molino
    - **Modelo**: Molino
        - **Escala**: 0.001
        - **Rotación**:
            - ◇ x: 0.1
            - ◇ y: 1
            - ◇ z: 0
    - **Paisaje**: renewable-4157213_1920.jpg
    - **Acciones**:
        - Nombre: girar
            - ◇ Animación: encender_led1
            - ◇ Repeticiones: 0

        - ◇ *Frame* final: último

- **Eventos**:
- **Medidas**:
    - Nombre: energia
        - ◇ Tipo: Numérico - Continuo
        - ◇ Unidades: MWh