Facultad
de
Ciencias

Solución a escala de una plataforma de gestión de una Ciudad Inteligente
(Smart City Platform Management Emulator)

Trabajo de Fin de Máster
para acceder al

Máster en Ingeniería Informática

Autor: David Martínez Gómez

Director: Michael González Harbour

Co-Director: _____

Junio – 2019

Agradecimientos

Quiero agradecer la ayuda y apoyo recibidos por parte de mi director de proyecto, así como de mis compañeros de la Oficina Técnica y Centro de demostraciones de Santander Smart City.

También agradecer a Mediavilla Fernández Informática S.L. de Reinosa por la cesión de materiales de proyección para el prototipo.

Resumen

Enmarcado en el proyecto de la Smart City se plantea la posibilidad de contar con una maqueta como herramienta de divulgación, flexible e intuitiva, que permita facilitar la comprensión de un sistema complejo como es una plataforma IoTData a gran escala. La solución estaría dirigida a personas con perfiles totalmente dispares, desde personas con conocimientos técnicos, a grupos de niños. El público objetivo de las presentaciones y formaciones comprende desde el ciudadano de a pie hasta el personal técnico. el personal técnico.

La maqueta ha de ser capaz de simular gran cantidad de servicios distintos, que no son estáticos, sino que están evolucionando rápidamente en el tiempo, y hacer llegar ese mensaje a cualquier tipo de público.

Se trata de un proyecto complejo para el que no se ha encontrado tecnología específica, y por tanto requiere de la realización de un prototipo para estudiar su capacidad y viabilidad antes de abordar el desarrollo completo, y precisamente ese será el objetivo de este proyecto, elaborar un prototipo de maqueta como estudio previo del proyecto final.

Se ha desarrollado una plataforma de gestión y simulación estructurada en capas, modular, escalable tanto horizontal como verticalmente y capaz incorporar nuevos servicios o adaptarse a cambios de manera transparente.

Como arquitectura general de la solución, se ha optado por una división de 4 capas, tanto a nivel de software como de hardware:

Capa IoT: Sensores y actuadores con hardware muy sencillo, conectados directamente a la maqueta. En esta capa encontraremos los sensores y actuadores (leds, motores, etc.) con pequeños controladores encargados de su gestión. Para este prototipo se ha utilizado Arduino a nivel de HW y C++ para el software.

Capa de Control IoT: Controladores/gateways de dispositivos IoT encargados de la gestión y las comunicaciones de red. Es la encargada de comunicar la capa IoT con el resto de la plataforma, es decir, los Arduinos con el resto de la red. Se ha elegido Raspberry a nivel de Hardware y C++ para el software.

Capa de contexto: Gestor de contexto (Context Broker) encargado de centralizar y distribuir las comunicaciones entre las distintas piezas de la arquitectura. Utilizaremos para esta capa el

Software Eclipse Mosquitto, que podrá correr sobre el mismo Hardware utilizado para otras capas, o sobre un hardware específico, según el caso.

Capa de Simulación: Software de simulación encargado de lanzar secuencias de simulación para los distintos dispositivos o simular dispositivos "inexistentes". También aporta capacidades multimedia a la plataforma. Se ha elegido Java para el software y se podrán usar múltiples dispositivos a nivel de hardware.

Como se irá explicando durante esta documentación, se ha construido una plataforma modular y fácilmente escalable, que necesita muy poca configuración y a un coste muy reducido. Aún con simulaciones sencillas se obtienen unos resultados muy llamativos que cumplen con creces los objetivos del proyecto. Se ha conseguido, por tanto, construir una plataforma viable y funcional que nos habilita para justificar las posibilidades de implantación del proyecto.

Palabras Clave

Smart City, Plataforma, IoT, Simulación, Maqueta.

Abstract

As part of the Smart City project, the idea of having a mock-up, as a flexible and intuitive dissemination tool, to facilitate understanding of a complex system such as a large-scale IoT/Big-Data platform has been considered. The solution would be designed for people with completely different profiles, from people with technical knowledge, to groups of children. The target audience for demonstrations and training, ranges from ordinary citizens, to technical staff.

The mock-up must be able to simulate a large number of different services, which are not static, but are quickly evolving over time, and get that message to any kind of public.

A full mock-up is a complex project for which no specific technology has been found, and therefore it is advisable to create a prototype mock-up to study its capabilities and viability before dealing with the full development. That will be the aim of this project, to develop a prototype mock-up as a first step for approval of the final project.

A management and simulation platform has been developed, structured in layers, modular, scalable (both horizontally and vertically), and capable of transparently adding new services or changes in existing ones.

A four-layer division has been chosen as the overall architecture of the system, both at the software and hardware levels:

IoT layer: It contains sensors and actuators with very simple hardware, directly connected to the mock-up. In this layer we can find sensors and actuators (leds, motors, etc.) with small controllers in charge of their management. For this prototype, Arduino has been used as controller at HW level and C++ for the software part.

IoT Control Layer: IoT device controllers/gateways in charge of communications and network management. This is the component that links the IoT layer with the rest of the platform, i.e, the Arduinos with the rest of the network. Raspberry Pi has been chosen as hardware and C++ as programing language to develop the software in this layer.

Context layer: A Context Broker is in charge of communications between the different parts of the architecture. We will use Eclipse Mosquitto as a software solution for this layer. It would run on the same hardware used for other layers, or on a specific hardware, depending on the case.

Simulation Layer: Simulation software is in charge of launching simulation sequences reproducing different devices, or simulating "non-existent" ones. It also provides multimedia capabilities to the system. Java has been chosen for the software development, and multiple devices can be used at the hardware level.

Based on this architecture, a modular and easily scalable platform has been built, which requires very few configurations, all this with a very low cost. Even with simple simulations, very impressive results are achieved, meeting the project objectives by far. We have therefore managed to build a functional platform that empowers us to implement the final project.

Keywords

Smart City, Platform, IoT, Simulation, Mock-up.

Índice General

1. Introducción

1.1 Ciudad inteligente o Smart City

El concepto de Smart City es relativamente nuevo, y por tanto aún está sujeto a continuos ajustes y modificaciones. Por esta razón es usual encontrarse definiciones muy genéricas y comerciales, centrándose más en los posibles efectos beneficiosos de implantar una ciudad inteligente que en la definición de la misma.

> *Ciudad inteligente [Smart City] es la visión holística de una ciudad que aplica las TIC para la mejora de la calidad de vida y la accesibilidad de sus habitantes y asegura un desarrollo sostenible económico, social y ambiental en la mejora permanente. Una ciudad inteligente permite a los ciudadanos interactuar con ella de forma multidisciplinar y se adapta en tiempo real a sus necesidades, de forma eficiente en calidad y costes, ofreciendo datos abiertos, soluciones y servicios orientados a los ciudadanos como personas, para resolver los efectos del crecimiento de las ciudades, en ámbitos públicos y privados, a través de la integración innovadora de infraestructuras con sistemas de gestión inteligente.*

[DelRivero2017]

Según *FIWARE*:

> *Making a city "smart" means turning your city into an ICT enabler for innovation, economic growth and well-being. By adopting common standards and information models, cities can achieve this transformation with minimum impact, merging forces to build an ecosystem where they can connect and collaborate. This enables the creation of interoperable and portable solutions that can be adapted and replicated for the needs of each city, building a sustainable Digital Single Market for Smart Cities.*

www.fiware.org [Fiware01]

Según *technopedia.com*:

> *A smart city is a designation given to a city that incorporates information and communication technologies (ICT) to enhance the quality and performance of urban services such as energy, transportation and utilities in order to reduce resource consumption, wastage and overall costs. The overarching aim of a smart city is to enhance the quality of living for its citizens through smart technology.*

www.technopedia.com [Technopedia01]

Esta falta de madurez en la definición del término, unido a que su uso en la mayoría de los casos es meramente político o de marketing, ha hecho que la definición de Smart City venga de forma recurrente con otros términos de moda adheridos, como son, sostenibilidad, ecología, calidad de vida, etc. No es que estos términos no tengan ninguna relación con las SmatCities, pero

desde luego, a mi forma de ver, no la definen, aunque son altamente efectivos a la hora de vender la idea.

Viendo una SmartCity, como una herramienta, por ejemplo, un martillo, si se me permite el símil, no es correcto, desde un punto de vista estricto, decir que un martillo sirva para sujetar cuadros, es la alcayata la que sujeta el cuadro, el martillo es una herramienta que sirve para golpear o clavar, con lo que puede servir para colocar la alcayata, no para sujetar el cuadro. Por tanto, sujetar cuadros, es solo un beneficio derivado de uno de sus múltiples usos.

De esta forma, aceptando la Smart City como una herramienta más para el correcto gobierno de una ciudad moderna, no me parece correcto definirla por medio de beneficios derivados de alguno de sus usos, como pueden ser la sostenibilidad, ecología o calidad de vida.

Una ciudad es un sistema enormemente complejo, en el que intervienen un número casi infinito de variables. Para mí, una ciudad inteligente es la que, para su gestión, intenta tomar consciencia de todas esas variables, midiéndolas, registrándolas y analizándolas en busca de relaciones y patrones que sirvan de apoyo en la toma de decisiones. Esto supone la implantación de sistemas de medición y sensorización, que producen un enorme volumen de datos, y requieren de la implantación de una plataforma cooperativa para gestionarlos. Para manejar este ingente volumen de datos se hace necesaria la utilización de tecnologías de la información, como Internet de las cosas, Big Data, aprendizaje automático (machine learning) o inteligencia artificial.

Por otra parte, la Smart City aparecería como un servicio horizontal, común y cooperativo para todos los servicios de ciudad, en contraposición a los servicios verticales (silos) que funcionarían de forma independiente y aislada unos de otros. Para su correcto funcionamiento, todo esto tiene que ir acompañado de un considerable aumento en el grado de madurez de todas las operaciones que forman parte del ecosistema de la ciudad, definición de procedimientos, eliminación de duplicidades, automatización de actividades, etc.
A diferencia del discurso político al que estamos acostumbrados, considero que la Smart City no es un fin en sí misma, sino una herramienta indispensable para la correcta gestión de una ciudad moderna.

1.2 Motivación

Una plataforma de ciudad inteligente es un proyecto de gran envergadura que requiere de la participación de infinidad de actores, tanto personal del Ayuntamiento, empresas adjudicatarias como ciudadanos y visitantes de la ciudad. Se trata de una cantidad enorme y heterogénea de actores, que interactuarán con la ciudad a distintos niveles, y con intereses muy dispares.

Se trata de un proyecto complejo y difícil de comprender por el público no técnico, sobre todo con la cantidad de ruido informativo y conceptos confusos que nos inundan, haciendo un uso puramente comercial e interesado de términos "*de moda*" entre los que últimamente se encuentra el término "Smart" y especialmente las Smart Cities, haciendo difícil a la gente de a pie distinguir entre la realidad de una ciudad inteligente y los conceptos de "ciencia ficción" que estamos acostumbrados a ver en los medios.

Para garantizar el éxito del proyecto de la Smart City, se hace necesaria una correcta divulgación y explicación del mismo, por este motivo se creó en Santander el Centro de Demostraciones encargado de divulgación del proyecto y al que progresivamente se va a dotar de herramientas que clarifiquen y refuercen el mensaje.

Llegados a este punto, una maqueta demostrativa de los principales servicios de la Ciudad inteligente, integrada en el Centro de demostraciones, supondría una herramienta vital y de mucho impacto a la hora de explicar el proyecto. Su funcionamiento no se limitará a la simulación, sino que se trata de una arquitectura a escala, que nos permitirá también utilizarla para las demostraciones y formaciones de personal técnico.

Si bien el grado de complejidad de una maqueta es sustancialmente inferior al que podemos encontrar en una ciudad real, se trata de un proyecto de suficiente envergadura como para exceder el ámbito de este trabajo de fin de máster. Dada la magnitud del proyecto, resulta aconsejable la realización de un estudio previo a su aprobación, centrado principalmente en un prototipo, que haga las veces de pequeña prueba de viabilidad y que sirva de base arquitectónica para la posible solución final. Este prototipo, que tendrá una envergadura más contenida, es sobre el que se centrará este trabajo fin de máster.

1.3 Objetivos del Proyecto

El objetivo final es construir un prototipo funcional de una maqueta de una ciudad inteligente, capaz de interactuar con el público y reproducir situaciones reales de una ciudad.

Durante la realización del prototipo se utilizarán recursos y técnicas aplicables al modelo final, de forma que sirva como base para la construcción y dimensionado del mismo.

La maqueta debe ser una herramienta capaz de apoyar la divulgación y formación en conceptos relacionados con la ciudad inteligente aportando un soporte físico y visual para la realización de demostraciones.

1.4 Alcance

- Implementar un **sistema empotrado** encargado de controlar el hardware de la maqueta, que puede estar compuesto de sensores, *(de temperatura, lumínicos, barreras de infrarrojos, etc.)* y actuadores *(motores, servos, luces y leds, etc.)*.
- Implementar **software de simulación**, que sea capaz de lanzar y controlar distintas simulaciones de actividades, susceptibles de ser controladas o monitorizadas por una plataforma Smart City. Ej.: Gestión de luminarias, residuos, tráfico, aparcamientos, etc.
- La solución debe ser **modular y escalable** en múltiples niveles, tanto vertical, cambiando dispositivos por otros con mayor capacidad, como horizontalmente, añadiendo nuevos dispositivos.
- Deben utilizarse **estándares**, tanto de comunicación (*MQTT [MQTT01]* o *NGSI [Fiware02]*), como para el modelo de datos (FIWARE Data Models), dado son una parte importante del mensaje que se desea transmitir.
- Tiene que ser posible cargar y lanzar distintas simulaciones de forma **determinista**.
- La maqueta tiene que ser capaz de recoger información y acciones procedentes de los usuarios participantes en la demostración.
- Construir un prototipo funcional, encargado de mostrar a grandes rasgos las posibilidades de la solución, su viabilidad y servir de soporte para determinar el alcance del proyecto final.

1.5 Estudio de la Situación Actual

La mayoría de las maquetas con fines demostrativos que podemos encontrar son sistemas específicos (ad-Hoc) diseñados para realizar un número limitado de operaciones concretas, que, para ser modificadas o ampliadas, requieren de un reanálisis y diseño a bajo nivel, que suele ser costoso. Esto choca frontalmente con la filosofía de IoT (Internet of Things), que es precisamente lo que se quiere demostrar, y produciría problemas a futuro puesto que una Ciudad inteligente es un sistema complejo y vivo, sujeto a constantes cambios, a los que un sistema "rígido" no sería capaz de adaptarse.

Ejemplos:

1.5.1 Mit Media Lab: CityScope – Andorra

Maqueta que simula la movilidad de las personas en la ciudad de Andorra basándose en datos obtenidos de los móviles.

- https://vimeo.com/235305199
- https://www.media.mit.edu/articles/cityscope-andorra-data-observatory-an-agent-based-visualization-on-tourism-patterns/

1.5.2 mOway Smart City

Pequeña maqueta modular, disponible para su venta en internet.

- https://www.youtube.com/watch?v=9XGOeYUN1Bs

1.6 Evaluación de Alternativas

1.6.1 Arquitectura de comunicaciones IoT

En el Internet de las cosas (IoT) nos encontramos con requisitos especiales que justifican la necesidad de buscar Arquitecturas y soluciones específicas. En resumen, vamos a encontrarnos con un gran número de dispositivos que no deberían ser interdependientes, comunicándose entre sí a través de una plataforma que debe ser fácilmente escalable.

Para satisfacer dichos requisitos, han surgido multitud de metodologías y arquitecturas, resumiremos a continuación las más destacadas para tomarlas como base para el estudio de nuestro proyecto.

1.6.1.1 Arquitecturas de Mensajería IoT

La solución más extendida para abordar la mensajería entre dispositivos IoT es la de delegar las comunicaciones en un servidor que actuará como centralita(router) y al que se suele denominar como Broker. Será pues este servidor el que contendrá el hardware de comunicaciones más costoso, y también será el encargado de la gestión de la mensajería, permitiendo así simplificar y abaratar los dispositivos IoT.

Para la gestión de la mensajería, el Broker podrá seguir distintos patrones de comunicación:

- **Patrón Publicador-Subscriptor**

  El patrón de comunicación Publicador-Subscriptor, o Publisher-Subscriber, consiste en que los dispositivos (publicadores) envían mensajes al Broker, los dispositivos también pueden pedirle al Broker o gestor de contexto, que les notifique de los nuevos mensajes, identificados por un id o categoría (subscriptores). De esta forma, cualquier dispositivo publicará un mensaje sin preocuparse de quienes son los destinatarios, y será el Broker el encargado de saber a quiénes se lo tiene que notificar.

- **Patrón Router de llamada a procedimientos remotos**

  El patrón Router Remoder Procedure Calls (rRPC) se basa, como su nombre indica en la ejecución remota de procedimientos. Un agente solicita la ejecución de un procedimiento a otro agente a través de un router, que le comunica el resultado al primer agente una vez que el procedimiento haya finalizado.

Independientemente del patrón elegido, el router, o bróker puede seguir dos filosofías a la hora de gestionar los mensajes:

- **Colas de mensajes**

En una cola de mensajes, el servidor asigna una cola a cada uno de sus clientes. De esta forma mantiene los mensajes recibidos hasta que son entregados correctamente, habitualmente en el mismo orden que llegaron.

- **Servicio de mensajes**
  En un servicio de mensajes puro, el servidor se limita a reenviar los mensajes recibidos, por lo que, si un cliente no está disponible, pierde todos los mensajes que le envíen en ese periodo de tiempo. Como es normal existen métodos para almacenar estos mensajes por otros medios si fuera necesario.
- **Datos Contextuales**
  A diferencia del servicio de mensajes puro, el servidor almacenará el último valor conocido para cada mensaje, diferenciándolo por medio de un identificador. Este valor puede ser consultado de forma activa en cualquier momento.

Dependiendo de las decisiones tomadas en los puntos anteriores tendremos que elegir el "idioma" que van a hablar nuestros dispositivos, es decir, el protocolo de comunicaciones. A continuación detallo algunos ejemplos:

- **MQTT, MQ Telemetry Transport:** Se trata de un protocolo Publicador-Subscriptor, sencillo y ligero, pensado para un gran número de clientes de forma simultánea.
- **AMQP, Advanced Message Queuing Protocol:** Protocolo Publicador-Subscriptor para colas de mensajes. Está diseñado para soluciones cooperativas de alto rendimiento y baja latencia. No es muy adecuado para dispositivos con bajos recursos.
- **WMQ, WebSphere MQ:** Es un protocolo de colas de mensajes desarrollado por IBM.
- **NGSI:** Protocolo de datos contextuales, que especifica todo el ciclo de vida de los mismos (Inserciones, actualizaciones, subscripciones).

1.6.2 Estudio de alternativas para nuestro proyecto

Al tener el prototipo una importante parte hardware, la solución estará muy centrada en la parte física, por lo que la decisión del hardware específico a utilizar será vital y será la base sobre la que se construirá el resto del proyecto. Se trata de una decisión a múltiples niveles ya que hay que decidir hardware de sensores, comunicaciones, servidores, etc. Por tanto, no se trata de una decisión que se pueda tomar aislada. Habrá que tomarla pensando en el tipo de plataforma que queremos construir, sus posibilidades, flexibilidad, capacidades, complejidad, etc.

Se redujo en un inicio el problema a dos opciones:

- Utilización de HW especializado (*el desplegado en la Santander Smart City*) específicamente diseñado para un fin, sensores de ciudad, servidores de la Smart City, etc.
- Utilización de HW de bajo coste y de uso común (*HW commodity*), como pueden ser *Arduino*, *Raspberry*, etc.

Otra decisión primordial será el corazón software de la plataforma, se ha optado por un patrón Publicador-Subscriptor y se evaluarán como alternativas la gestión de datos contextuales con el protocolo NGSI frente a un servicio de mensajería puro con el protocolo *MQTT*. Se han elegido estos dos protocolos por estar presentes en la plataforma real a simular.

Estos estándares no son dos opciones opuestas, o excluyentes, ya que es habitual utilizar *MQTT* dentro de una plataforma *FIWARE*, y ambas comparten filosofía y gran parte de los objetivos. Su principal diferencia es el tamaño y complejidad de las soluciones resultantes de su implementación.

A continuación, se describen estas alternativas con mayor detalle y se justifica la elección realizada.

1.6.3 Alternativas de hardware

Plataforma con hardware especializado

Aunque ya a priori se puede sospechar que esta plataforma no va a ser la opción más atractiva, al tratarse de hardware diseñado para cumplir con requisitos de carga y rendimiento muy superiores a los que vamos a encontrar en una maqueta, conviene realizar un estudio antes de descartarla. No será un esfuerzo inútil ya que al fin y al cabo en el peor de los casos se trata de un estudio del entorno a simular. Y si bien no parece razonable reproducir toda la infraestructura real, es muy posible que resultase interesante utilizar parte de ella en determinados niveles o capas, o simplemente integrar la maqueta como una parte más de la plataforma, aportando datos y estadísticas.

Como se ha mencionado anteriormente, hay que abordar esta decisión en varios niveles, fuertemente relacionados:

- *Sensorización IoT*

Cuando se observan los sensores reales desplegados en la ciudad de Santander, en los contenedores de basura, en las luminarias, los sensores de tráfico, etc.; nos damos cuenta de que su tamaño determinaría de forma dramática las posibilidades de escalado de la maqueta. Se trata de equipos con un tamaño considerable, que forzarían a que las demostraciones de los mismos estuvieran implementadas en una escala bastante aproximada a la real. Para eso ya están habilitadas varias zonas del Centro de Demostraciones de la Smart City, y estos sensores quedan por tanto inmediatamente descartados para la implementación de esta solución.

Por si esto fuera poco, las soluciones implementadas en la plataforma real están compuestas por sistemas totalmente heterogéneos, fruto de decisiones tomadas desde distintos proyectos, lanzados por distintos servicios, y que utilizan protocolos y sistemas de interconexión diversos, que introducirían una inmensa complejidad, tanto en el diseño de la solución, como en su implementación.

Otra desventaja es que generan dependencia de un proveedor (*vendor lock*). Quedaríamos pues, a merced de múltiples fabricantes y empresas a la hora de adquirir repuestos o actualizaciones. Algo que no parece sostenible a largo plazo.

- *Gestión IoT y comunicaciones*

Para la gestión y comunicaciones de los sensores con capas superiores de la Smart City, se utilizan pasarelas (gateways) y sistemas de comunicación, que habitualmente emplean algún tipo de comunicación inalámbrica, como: GSM, Zigbee, Wifi, 3G, 4G…

Si bien estarían disponibles los dispositivos de repuesto utilizados en la plataforma de ciudad, no parecen soluciones razonables dada la escala que se pretende que tenga la solución, y la escasa distancia a la que se encontrarán sus distintos componentes.

También nos encontramos con el problema de que en la plataforma real las soluciones de gestión y comunicación implementadas son muy heterogéneas, debido a la gestión aislada de algunos servicios y la variedad de proveedores y empresas adjudicatarias. Descartamos por tanto también la utilización de este tipo de hardware para esta capa.

- *Servidores*

Entramos en la capa de hardware más relacionada con el procesado de información, que está conformada por distintos servidores, Linux en su mayoría, dentro de la red del Ayuntamiento.

Este modo de despliegue supone implementar comunicaciones TCP, e integrarse dentro de las políticas de seguridad, que, si bien aportan más robustez y funcionalidad, no son necesarias dado los requisitos y el alcance del proyecto, solo aportan mayor trabajo de configuración y despliegue, disparando los costes de mantenimiento, tanto a nivel económico, como de personal.

Plataforma Commodity

- *Sensorización IoT*

Otra posibilidad, es utilizar la plataforma *Arduino* para controlar los diferentes sensores y actuadores dentro de la maqueta.

Se trata de una plataforma respaldada por una inmensa comunidad en internet, que aporta librerías, y constantes actualizaciones. Otra ventaja innegable es económica, al tratarse de hardware con un inmenso mercado, los costes se han reducido a niveles irrisorios, y el acceso a repuestos está garantizado de forma rápida e ilimitada, sin atarse a ninguna marca o fabricante.

Se trata pues de implementar pequeños circuitos basados en Arduino, capaces de gestionar soluciones de luminarias, contenedores de basura, plazas de aparcamiento, etc.

Luminarias: Empleando leds y una foto-resistencia se puede implementar un sistema de gestión de luminarias.

Gestión de Residuos: Controlado por un *Arduino*, mediante una barrera de infrarrojos controlamos el nivel de llenado de un pequeño contenedor y con un servo simulamos su vaciado mediante la apertura de una compuerta inferior, bien para simular la descarga mediante un camión o un sistema de residuos soterrado.

Gestión de Tráfico: Mediante interruptores implementados con interruptores magnéticos, e imanes se pueden simular plazas de aparcamiento, o la detección de vehículos en posiciones estratégicas que permitirán enriquecer las simulaciones anteriores.

Los dispositivos Arduino tienen ciertas limitaciones, en cuanto a número de entradas, salidas y también en cuanto al amperaje que son capaces de aportar sin la utilización de circuitos más complejos. Pero, por otra parte, su reducido tamaño y bajo coste, unido al diseño con vocación modular, nos permite escalar la plataforma horizontalmente, es decir, se puede ampliar la capacidad del sistema añadiendo nuevos dispositivos y consiguiendo así fácilmente mayor número de entradas y salidas manteniendo una solución muy modular.

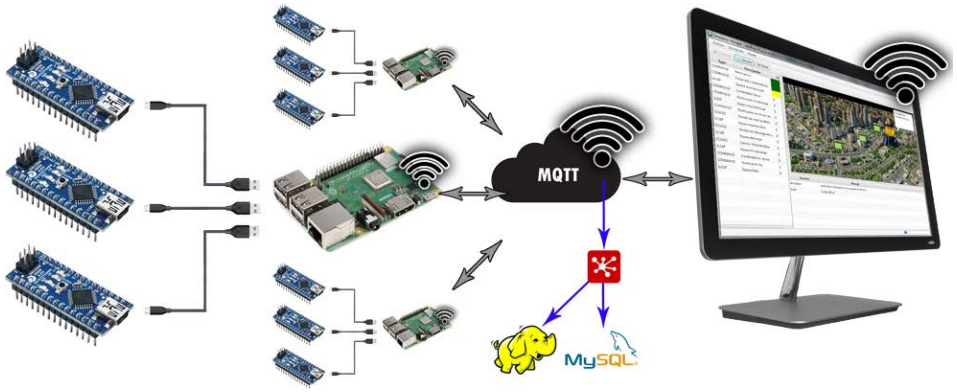
Como es lógico, también es posible, si fuera necesario, un escalado vertical, sustituyendo dispositivos individuales por otros equivalentes con más recursos y mayor capacidad de cómputo.

- *Gestión IoT y comunicaciones*

Con los sensores Arduino tendríamos resuelta la sensorización y la parte de la plataforma que requiera tiempo real.

Lógicamente es necesario que todos esos dispositivos se comuniquen entre sí, y para ello hay disponibles en el mercado módulos de comunicación para cualquier tipo de Arduino, y utilizando casi cualquier tecnología de comunicación a nivel físico, Ethernet, Wifi, bluethooth, bus CAN, …

Se ha optado por utilizar un controlador de dispositivos serie, conectando los *Arduino* por USB, implementado en un procesador de bajo coste tipo *Raspberry* y basado en protocolo *MQTT* sobre TCP, ya que se trata de un estándar muy extendido en plataformas IoT, con una amplia comunidad aportando soporte y del que se pueden encontrar multitud de librerías y software libre.



Esquema simple de arquitectura del sistema

En la figura, podemos ver la arquitectura propuesta, los controladores IoT se conectan a las Raspberry Pi mediante USB, esta se conecta con el gestor de contexto MQTT que se encarga de distribuir los datos al PC de gestión de simulaciones, o a otros servidores para su almacenamiento, por ejemplo, una base de datos MySQL u otro gestor de contexto.

Los distintos controladores IoT basados en *Arduino* se comunicarán mediante un simple cable USB con la *Raspberry PI*, que actuará como pasarela de comunicaciones, y que nos evitará al mismo tiempo alimentar a los Arduinos con fuentes de alimentación externas. Este montaje permite que los diferentes controladores se comuniquen con el resto de componentes de la maqueta mediante *MQTT*, y sin tener que implementar costosos interfaces de comunicaciones. La desventaja es que tendremos que implementar todo el software de gestión de módulos y comunicaciones para las Raspberry PI, pero evitaremos tener que hacerlo, uno por uno, en todos los *Arduino*.

Por tanto, el controlador *Raspberry PI*, que actuará como una centralita o Gateway, se encargará de detectar automáticamente los módulos Arduino que tiene conectados, comunicarse con ellos mediante un simple *MQTT* por puerto USB, y gestionar las comunicaciones más complejas, bien vía ethernet, bien vía Wifi, con un gestor de contexto MQTT, como por ejemplo Eclipse Mosquitto, que hará las veces de "centralita" de mensajes para toda la maqueta.

El funcionamiento de los Arduino es por tanto plug&play, basta con conectarlos a la *Raspberry*, y esta los detectará y configurará de forma automática con el resto de la plataforma. Simplemente conectando el *Arduino* con el USB este pasa a ser 100% funcional dentro de la plataforma

Al construirse como un gateway, la *Raspberry PI* también es escalable horizontalmente, es decir, se pueden conectar a la plataforma tantos controladores *Raspberry* como se quiera, de una forma transparente y sin necesidad de ninguna configuración adicional.

- *Servidores*

Debido a la simplicidad de la plataforma commodity, no será necesario el despliegue de máquinas especiales, siendo suficiente un PC que hará las veces de servidor *MQTT* y de gestor de las simulaciones.

1.6.4 Alternativas a nivel de Software

Plataforma Fiware

Se trata de una plataforma de código abierto para soluciones Smart promovida por la Unión Europea, que se está constituyendo como estándar principal para las Smart Cities.

Según su propia definición, *FIWARE* es:

> 
>
> *FIWARE is a curated framework of open source platform components to accelerate the development of* ***Smart Solutions***

www.fiware.org [Fiware01]

La pieza central de *FIWARE* es su gestor de contexto, *Orion Context Broker*, solución de código abierto desarrollada por Telefónica en C++. Este gestor mantiene en una base de datos *no relacional*, una copia del último estado conocido de lo que llama entidades, que no son más que sensores, mediciones, dispositivos, indicadores de rendimiento (Kpi), etc. También aporta una serie de funcionalidades como son, consultas filtradas, Geolocalización (Geo:Json), paginación, capacidad de gestión independiente para múltiples clientes (Multi-tenancy), asociación jerárquica con otros Context Brokers (Federación), etc.

Para las comunicaciones, Orion utiliza el protocolo de comunicaciones *NGSI*, también de FIWARE, pensado para manejar información contextual, para lo que define la estructura de las comunicaciones, así como los modelos de datos, basados en el *Fiware Data Model.*

Dado que el Context Broker, solo entiende *NGSI*, habitualmente se hace necesaria la utilización de otra pieza del ecosistema *FIWARE*, los *IoTAgents*, que se encargan de gestionar todo tipo de sensores, recibir sus comunicaciones y convertirlas a *NGSI* para ser enviadas al Context Broker. Por ejemplo, un *IoTAgent MQTT* se encargaría de gestionar sensores *MQTT* y de realizar las conversiones necesarias con *NGSI* para garantizar las comunicaciones en ambos sentidos.

Esto hace necesario, aprovisionar los sensores en el *IoTAgent*, y a su vez, aprovisionar los agentes en el Context Broker.

Una vez le lleguen los datos de una entidad al Context Broker, este almacena su estado, y reenvía el mismo a todos los actores que lo hayan solicitado mediante subscripción, enviando a cada uno de ellos una notificación *NGSI*. Estas notificaciones, por cuestiones de rendimiento, no son reenviadas en caso de error, con lo que, si se quiere garantizar la consistencia de los datos, se hace necesaria la utilización de otra pieza de la plataforma que aporte esta funcionalidad. Se trataría de *Cygnus*, que desplegado en el mismo entorno de alta disponibilidad que el CB, y mediante colas basadas en *Apache Flume*, no solo nos garantiza el reenvío de las notificaciones en caso de error, sino que dispone de diferentes sumideros de datos (sinks) que son los encargados de almacenar estas notificaciones de forma persistente en diferentes plataformas, como por ejemplo: *SQL*, *HDFS*, *MongoDB*, *Kafka*, *CKAN*, *PostgreSQL*, o incluso en otro Context Broker.

Aun siendo una visión básica y resumida de lo que sería una plataforma Fiware, se pueden entrever las implicaciones que conllevaría el despliegue y configuración de una plataforma de este tipo, que implicaría la utilización de múltiples servidores, así como el aprovisionamiento de sensores y comunicaciones a varios niveles. Una complejidad a todas luces excesiva para el proyecto que nos ocupa, con lo que se ha descartado para el uso como base de la plataforma. Ahora bien, dado que es la plataforma implementada en Santander, no solo se utilizará como referencia para la simulación, sino que se podrá utilizar para el envío de estadísticas de uso de la maqueta, o notificación de errores como si nuestra maqueta fuera un servicio o sensor de ciudad cualquiera.

Plataforma basada en MQTT

Se ha optado por una solución menos compleja, montada en torno a un gestor de contexto basado en *MQTT*, en concreto *Apache Mosquitto*, que a diferencia de *Orion*, es un simple gateway o centralita software, encargado de la transmisión y distribución de mensajes *MQTT* mediante la subscripción a categorías de mensajes o *topics*. No requiere de ningún tipo de aprovisionamiento o configuración con lo que no supone una sobrecarga complejidad a nuestro proyecto.

A este gestor de contexto, se conectarán mediante subscripción las Raspberry, el software de simulación, y las operaciones de extracción, transformación y carga de datos, más conocidas como *ETL*, que serán desarrolladas con la herramienta de código abierto de Hitachi, *Pentaho Data Integration*, y serán las encargadas de recopilar y enviar estadísticas al Context Broker real de la ciudad, así como de almacenar de forma persistente determinados mensajes en una base de datos, si se considera necesario.

Tanto las *Raspberry* como el software de simulación utilizan la librería *PAHO*, también de *Apache*, para gestionar las conexiones al servidor *MQTT* de forma asíncrona. Se ha utilizado la librería de C en el caso de las Raspberry y la versión de Java para el software de simulación.

Software de control Raspberry

Para el software de control de dispositivos *Arduino*, implementado en las *Raspberry*, se ha decidido utilizar C++, teniendo en cuenta que se va a ejecutar en un entorno Linux, que es el lenguaje nativo de la librería *PAHO*, y que ofrece las mejores posibilidades de optimización de código, permitiendo además multiproceso y tiempo real laxo mediante el uso de los recursos del siestema operativo.

Se ha optado por utilizar la versión *Community* de *Visual Studio 2017* como entorno de desarrollo, que además de ser gratuito, viene preparado para la compilación y depuración cruzada, compilando y ejecutando directamente en la plataforma final (Raspberry Pi) mientras desarrollamos cómodamente desde un PC, lo que ha facilitado enormemente el trabajo realizado para realizar esta pieza de la plataforma.

Software de simulación

La pieza software más visual, y que estará en contacto directo con un usuario humano es el software de simulación, capaz de lanzar simulaciones sobre la plataforma. Se ha elegido para su desarrollo Java, por la simplicidad y sus capacidades multiplataforma, utilizando, como se ha

mencionado anteriormente, la librería de desarrollo *PAHO* para las comunicaciones *MQTT* y *JavaFx* para la Interfaz de usuario. La elección de *JavaFX*, como se explicará más adelante en profundidad, está principalmente debida a la enorme capacidad que aporta a Java a la hora de diseñar interfaces gráficas de usuario.

Esta pieza de la plataforma será también la encargada de otorgar capacidades multimedia a la plataforma, como pueden ser la emisión de música y/o efectos de sonido Fx, vídeo, etc.

Como es habitual cuando se trata de desarrollo Java, se ha elegido Eclipse como plataforma de desarrollo.

1.6.5 Resumen de alternativas

A modo de sumario, a nivel de hardware se ha optado por una plataforma montada sobre hardware de uso común, que simplifica la arquitectura de la plataforma abaratando los costes y también facilita el mantenimiento y la obtención de repuestos. Entre todos los dispositivos disponibles en el mercado se ha optado por controladores *Arduino Nano* y *Raspberry PI 3B+,* que vienen respaldados por una amplia comunidad.

A nivel de Software, se ha elegido montar la plataforma entorno a un gestor de contexto *MQTT*, concretamente ***Eclipse Mosquitto***. Para la parte de la plataforma encargada del control de la maqueta se ha optado por el lenguaje nativo en los ***Arduino Nano***, basado en **C++**, dada la escasa complejidad necesaria para los algoritmos implementados en estos módulos. En el caso de la ***Raspberry Pi*** también hemos elegido **C++** junto a la librería ***Eclipse Paho***, encargada de las comunicaciones *MQTT*.

Para la parte del simulador se ha optado por la utilización de **Java** + ***JavaFx***. Se ha tenido en cuenta la existencia de un amplio catálogo de herramientas de desarrollo libres y sus grandes capacidades multiplataforma.

2. Análisis

2.1 Definición del Sistema

Como ya se ha mencionado el Alcance del sistema se limita al desarrollo de un prototipo para el estudio de viabilidad y posibilidades de realización de una maqueta de demostraciones de proporciones y complejidad sustancialmente mayores.

Con este objetivo, se van a realizar tres pequeños casos de uso, que se corresponderán con dos módulos hardware y sus correspondientes simulaciones. Para este efecto se han elegido los servicios de alumbrado, estacionamiento y basuras.

2.2 Requisitos del Sistema

2.2.1 Obtención de los Requisitos del Sistema

Código	Nombre Requisito	Descripción del Requisito
R1.1	Ejecutar Simulaciones	La plataforma tiene que ser capaz de ejecutar simulaciones de forma determinista
R1.2	Control de dispositivos on/off	El sistema debe ser capaz de controlar el estado de dispositivos binarios (encendido/apagado), como pueden ser leds, ventiladores, bombas de agua, etc.
R1.3	Control de dispositivos complejos	La plataforma debe ser capaz de controlar dispositivos más complejos, como motores, servo-motores, etc.
R1.4	Sensores	Se podrán recibir datos de sensores, tanto digitales como analógicos.
R1.5	Música y sonidos	El sistema tiene que ser capaz de emitir efectos de sonido y/o música como parte de la simulación.
R1.6	Vídeo	Se han de poder emitir vídeos como parte de la simulación, bien para ser mostrados en un monitor, o bien para ser proyectados directamente sobre la maqueta
R1.7	Representación mapa	El software debe ser capaz de representar el estado de la simulación en un mapa de la maqueta, total o parcial.
R1.8	Tiempo de respuesta	Si bien no se trata de un sistema de tiempo real en el que los tiempos de respuesta son críticos, sí que es importante que exista una sensación de respuesta instantánea durante las demostraciones.

		Por tanto, los plazos serán no estrictos y se puede considerar el sistema como de tiempo real laxo con tiempos de respuesta muy reducidos.

Requisitos tecnológicos

Código	Nombre Requisito	Descripción del Requisito
R1.1	Tipo de arquitectura	La arquitectura debe seguir un patrón similar al utilizado en la plataforma real de la Santander Smart City. Esto nos permitirá utilizar la propia arquitectura de la plataforma para realizar demostraciones para un público con un perfil técnico.
R1.2	Commodity hardware	Se debe utilizar hardware de uso general, lo que reducirá los costes y facilitará la obtención de repuestos.

2.2.2 Identificación de Actores del Sistema

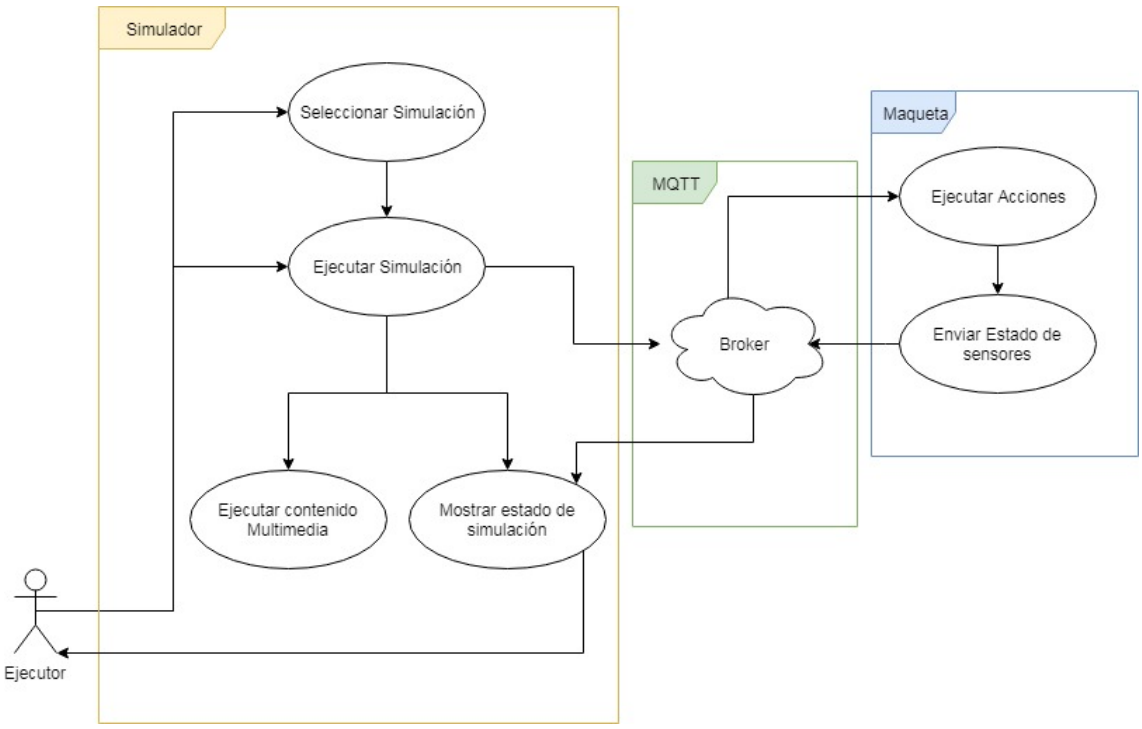
- Administrador de Simulaciones: Se trata del usuario con perfil técnico, encargado de diseñar las simulaciones. Lo incluimos aquí ya que está previsto que en la solución final el software de simulación contenga una herramienta gráfica que facilite estas tareas, pero en el ámbito del prototipo simplemente sería el encargado de generar los ficheros JSON *[JSON01]* que definen una simulación.
- Ejecutor de simulaciones: Se trataría de la persona encargada de ejecutar y presentar las simulaciones. Su perfil sería comercial o de formación.
- Público: Como ya se ha mencionado el público objetivo de las simulaciones abarca casi todo el espectro de población, en cuanto a edades y nivel de conocimiento técnico.

2.2.3 Especificación de Casos de Uso

Caso de uso principal

El caso de uso principal, el que describe de forma genérica el funcionamiento esperado de la plataforma, es sencillo de explicar:

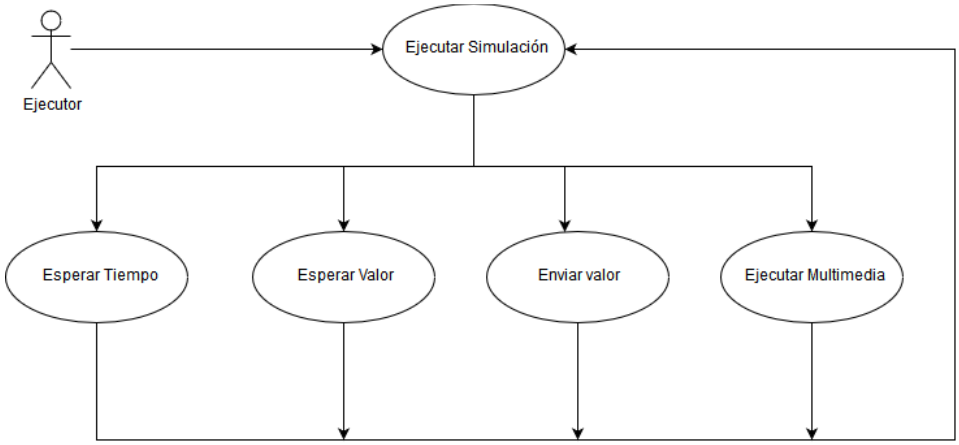
Un usuario Ejecutor de Simulaciones carga un fichero de simulación, y lo ejecuta.



Caso de uso principal

Ejecutar Simulación

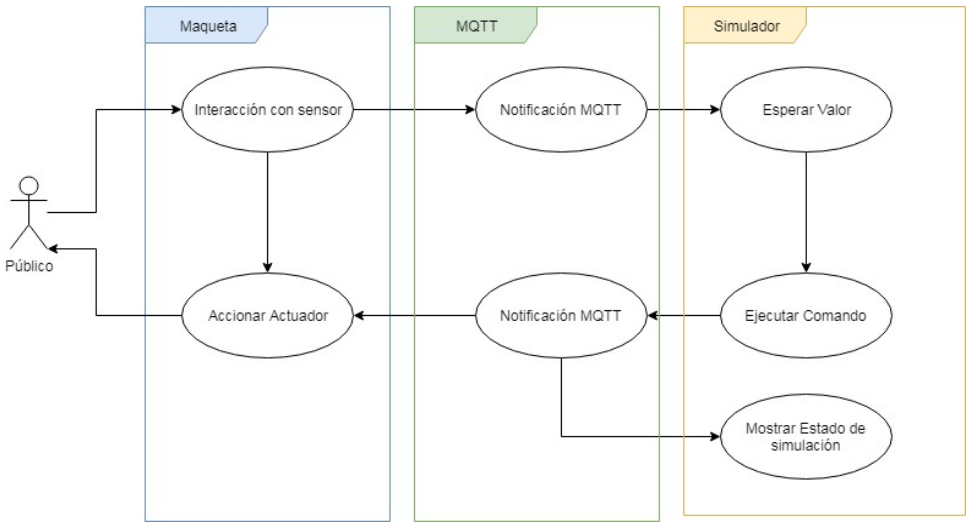
Este caso de uso detalla en que consiste el estado *Ejecutar Simulación* del caso de uso principal, y forma parte íntegramente del simulador. Básicamente el ejecutor lanza una secuencia de diferentes tipos de acciones predefinidas que conforman una simulación.



Ejecutar simulación.

Interacción con el público

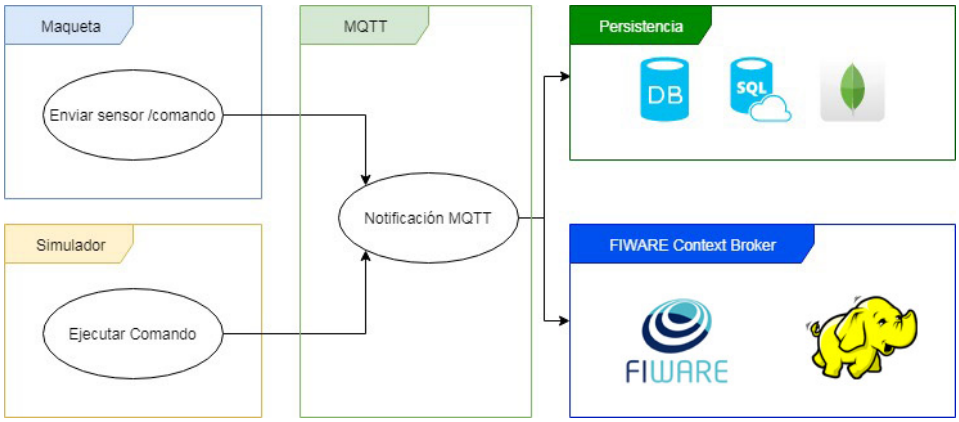
Algunas de las simulaciones implicarán interacciones con el público o con la persona encargada de realizar la presentación. Este caso de uso es el encargado de describir dicha interacción.



Interacción del público.

Persistencia de datos

El almacenamiento persistente de datos puede provenir de datos obtenidos directamente desde la maqueta o bien de datos provenientes del simulador, y deberá poderse hacer tanto en Base de Datos como integrándose con el Gestor de Contexto de la Smart City real.



Persistencia de datos.

2.3 Especificación del Plan de Pruebas

Al tratarse de un prototipo no se ha elaborado un plan de pruebas de integración continua, si bien sí se han implementado pruebas unitarias automatizadas, al considerarlas básicas y obligatorias en cualquier proyecto de software. El resto de las pruebas se han realizado de forma manual.

Se han contemplado los siguientes tipos de pruebas:

- **Pruebas Unitarias**: Una prueba unitaria es una forma de probar el correcto funcionamiento de un módulo de código o clase, que cumple con una función concreta. Esto sirve para asegurar que cada uno de los módulos funcione correctamente por separado, y se ejecuta de forma automática cada vez que se compila la aplicación, lo que nos permite detectar casi

al instante errores inesperados producidos por cambios en el código en otras partes del software.

Se han utilizado test unitarios utilizando la librería Java *JUnit* para el Gestor de simulaciones y un proyecto de Test unitario de Visual Studio para el código C++ de la Raspberry.

- **Pruebas de Integración**: Las pruebas de integración comprenden verificaciones asociadas a grupos de componentes, comprobando que éstos funcionan correctamente cuando son ensamblados para cumplir con una función concreta. Estas pruebas se han realizado de forma manual, durante el desarrollo y los despliegues, probando las integraciones a alto nivel entre:
    - Arduinos y Raspberry: Comunicación USB, detección de módulos, protocolo *MQTT*, etc.
    - Raspberry con el servidor *MQTT*: Comprobación de la comunicación en ambos sentidos.
    - Simulador con el servidor *MQTT*: como en el caso anterior, en ambos sentidos.

  Si el proyecto sigue adelante, hay que automatizar todas estas pruebas en la medida de lo posible.
- **Pruebas del sistema**: Las pruebas del sistema son pruebas de integración de todos los componentes que componen el sistema final, y que sus relaciones con otros sistemas que necesite son correctas, verificando así que las especificaciones funcionales y técnicas se cumplen.

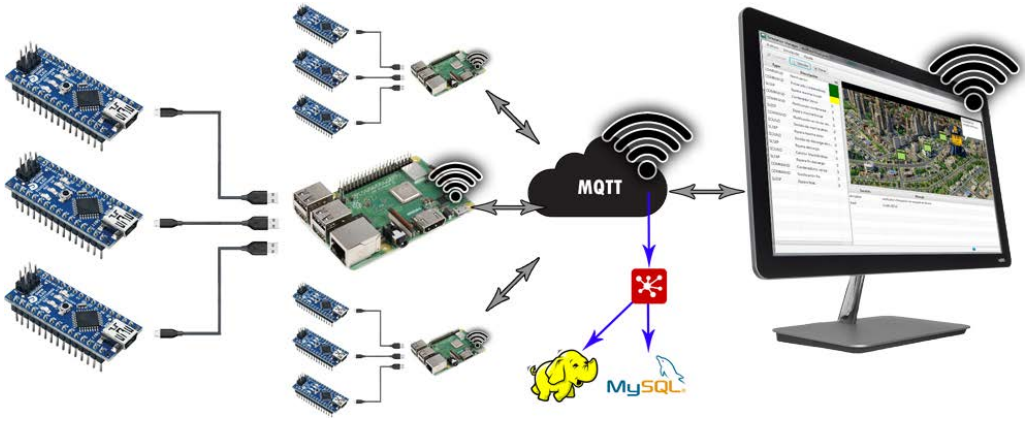
  Las pruebas de sistema (end to end) también se han realizado de forma manual, verificando que se cumplen todos los requisitos del sistema.

3. Diseño del Sistema

3.1 Arquitectura del Sistema

3.1.1 Arquitectura Hardware

Como se ha comentado con anterioridad la plataforma de control de la maqueta va a estar montada sobre tres niveles Hardware.



Esquema simple de arquitectura del sistema

Los tres niveles son:

- **Controladores IoT:** Implementado mediante *Arduinos Nano*, conectados y alimentados mediante cables USB.
- **Gestión IoT y comunicaciones:** Implementado mediante *Raspberry PI* (una o varias), y encargado de la comunicación entre la Capa IoT y el resto de la plataforma mediante un gestor de contexto *MQTT*, El funcionamiento de esta capa es transparente para las otras dos, y las *Raspberry PI* detectan de forma automática los módulos Arduinos que tienen conectados, con lo que aparte de la conexión *MQTT* de la propia *Raspberry PI*, no es necesaria ninguna reconfiguración de la plataforma si queremos sustituir o agregar nuevas *Raspberry PI*.
- **Equipo de gestión:** Se trata de un PC convencional encargado de la ejecución y monitorización de las simulaciones.

La pieza software encargada de la gestión de contexto *MQTT*, puede ser ejecutada tanto en la Raspberry como en el equipo de gestión, según las necesidades y conveniencia en cada caso. En nuestro caso se ha elegido instalarlo en el PC de simulación, al ser este equipo una pieza común a todas las configuraciones necesarias durante el desarrollo, es decir, vamos a necesitar el simulador funcionando sin la parte IoT, pero no vamos a necesitar que la parte IoT funcione sin el PC de simulación.

Como se puede ver en el gráfico anterior, a nivel de hardware, la plataforma es escalable tanto horizontalmente, añadiendo más Arduinos, más dispositivos Raspberry o incluso más dispositivos de simulación, como verticalmente, cambiando cualquiera de los dispositivos mencionados por otros con mayor capacidad y potencia.

Podemos crecer verticalmente:

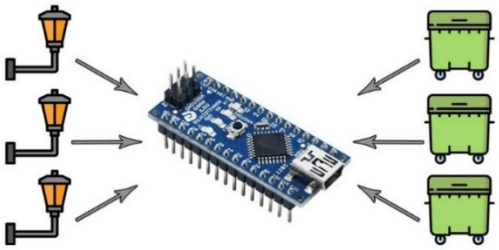
- **Controladores IoT:** Cambiando los Arduino por controladores más potentes, con más memoria, con más patillas de entrada/salida, etc.
- **Gestión IoT y comunicaciones:** Cambiando las *Raspberry PI* por modelos más potentes, con mejor procesador, con más puertos USB, tarjetas de red, etc.
- **Equipo de gestión:** Mismo caso de los anteriores, el equipo de simulación se puede sustituir por un equipo con más potencia.

Podemos crecer horizontalmente:

- **Controladores IoT:** Añadiendo módulos Arduino a los ya existentes, encargados de nuevos sensores y/o actuadores.
- **Gestión IoT y comunicaciones:** Añadir nuevas *Raspberry PI* para dar soporte a nuevos módulos Arduino, o para gestionar parte de los Arduino de otra Raspberry PI sobrecargada. Ninguna de estas ampliaciones requiere ninguna configuración ni en los Arduino ni en el gestor de contexto *MQTT* ni en el simulador.
- **Equipo de gestión:** También se pueden tener varios equipos de simulación funcionando de forma simultánea, por ejemplo, un equipo encargado de la visualización del estado de la maqueta y otro para el operario que lanza y gestiona las simulaciones. Otro ejemplo sería tener varios equipos mostrando simulaciones de distintas partes de la maqueta, por ejemplo 2 equipos, uno para el servicio de basuras y otro para el de alumbrado.

Arduino

Se han utilizado módulos *Arduino Nano* para la simulación al considerarse óptimo por su relación entre tamaño y número de pines I/O.

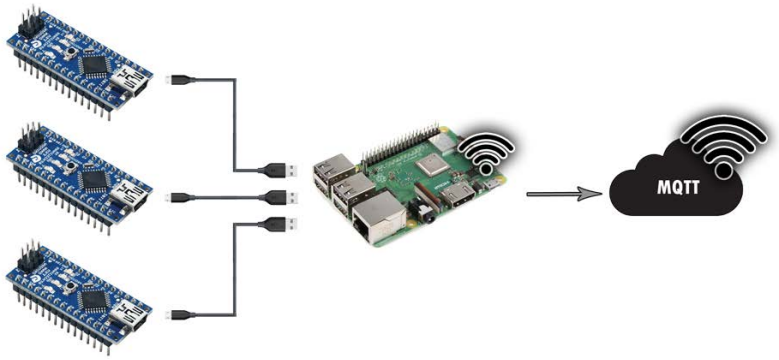


Módulo Arduino Nano

En la mayoría de los módulos será suficiente con la utilización de los pines digitales, unos 12 a efectos prácticos, cada uno capaz de proporcionar 40mA, y 6 analógicos, y un consumo total de todos los dispositivos conectados no puede superar los 500mA.

Raspberry

La *Raspberry* está conectada mediante USB a los módulos *Arduino* que controla, actuando como una especie de Router que permite comunicarlos con el gestor de contexto *MQTT*.



Módulo Raspberry

En la imagen anterior se muestra la conexión entre *Raspberry* y *MQTT* utilizando una red Wifi, pero el servidor *MQTT* podría estar en la propia *Raspberry*, y la conexión con el resto de la plataforma podría ser mediante cable ethernet, o cualquier otro tipo conexión TCP/IP.

Persistencia

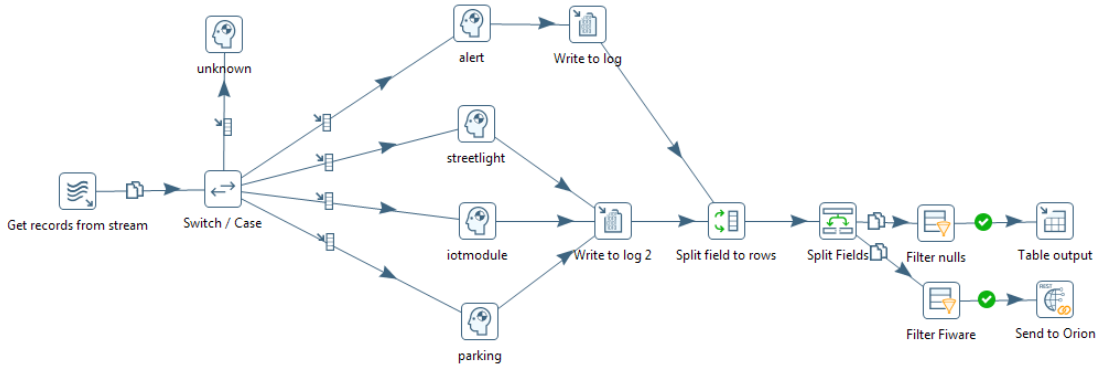
Siguiendo la filosofía de las Smart City, la maqueta también será capaz de recopilar información, persistir datos y enviarlos a la plataforma real de la Smart City como si un sensor de ciudad más se tratara, y así poder ser utilizados en un posterior análisis.

Datos interesantes que se podrían recopilar:

- Tiempos de utilización.
- Frecuencia de utilización de las distintas simulaciones, o incluso de los diferentes componentes de la maqueta.
- Estadísticas y patrones de interacción del público con la maqueta, lo que nos puede dar información no solo sobre el diseño y uso de la maqueta, sino también sobre los ciudadanos, sus preocupaciones y su manera de entender la Smart City.
- Registro de alertas y/o errores en caso de producirse.

Para el envío de datos desde la maqueta al gestor de contexto de la ciudad de Santander, se han utilizado las nuevas capacidades de transmisión (streaming) de datos de la herramienta de ETL *(extracción, transformación y carga de datos)* de *Pentaho BI (conjunto de herramientas de Business Inteligence).*

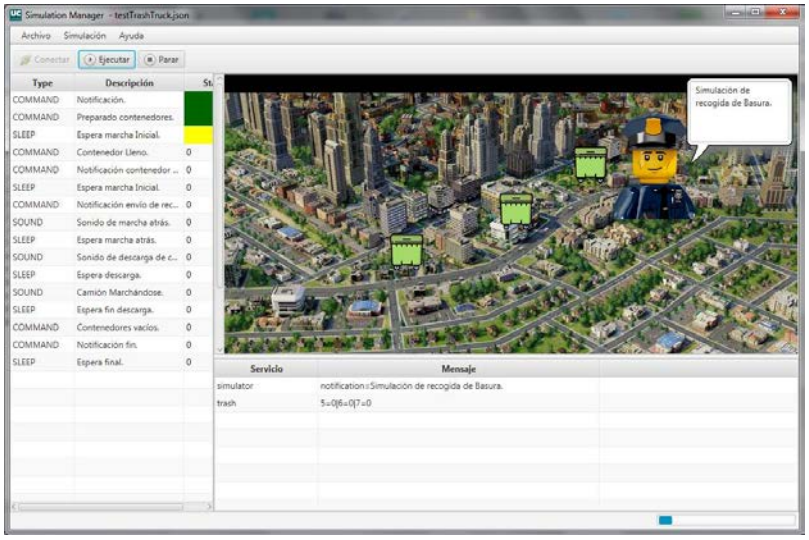
En su versión 8.1 *Pentaho Data Integration*, también conocida como *Kettle*, introduce entre sus nuevas características unos nuevos estados capaces de levantar un socket para escuchar los mensajes que lleguen por la red, y que serán utilizados como fuente de datos de la transformación ETL, transformándolos y guardándolos al vuelo, sin necesidad de ejecuciones periódicas. *Kettle* ya tiene un estado específico para comunicaciones *MQTT*, que nos permite realizar de forma nativa, tanto recepción (suscripciones), como envío de datos.



Transformación Pentaho Data Integration

En la imagen anterior se puede ver la transformación (*ETL de Kettle*) que se ha implementado para permanecer subscrita al servidor *MQTT* y encargarse de enviar a *Orion Context Broker* y *MySQL* los mensajes deseados.

Simulador



Interfaz de usuario del Simulador

El software de gestión de simulaciones es el encargado de lanzar las secuencias de simulación, monitorizar el estado de la maqueta y reproducir el contenido multimedia. Se trata de una aplicación construida en *JavaFx* sobre un proyecto Java, que implementa la lógica de negocio. Toda la arquitectura está pensada para realizar comunicaciones asíncronas, utilizando múltiples hilos de ejecución.

Se trata de una Herramienta pensada para ser utilizada por el usuario ejecutor, pero que, gracias a su arquitectura, modular y basada en el patrón modelo-vista-controlador (MVC) *[MVC01]* , sería fácilmente transformable en una o varias herramientas para el público.

Se ha elegido Java + *JavaFx* por ser multiplataforma, gratuitos y contar con una amplia comunidad que facilita tanto el desarrollo como el mantenimiento.

3.1.2 Diseño de Clases

Al tratarse de un sistema heterogéneo en lo que a lenguajes de programación y herramientas de desarrollo se refiere, se ha utilizado en cada caso la herramienta de generación de documentación adecuada para permitir la correcta gestión de los diagramas de clases durante el ciclo de vida del software, dando como resultado diagramas y estilos distintos, pero que nos aseguran una correcta correspondencia entre la documentación y la última versión de software.

Arduino

Para la implementación de los algoritmos de control de sensores y dispositivos, dada su simplicidad y la necesidad de obtener una baja huella de memoria, se ha utilizado programación estructurada tradicional, es decir no se ha utilizado programación orientada a objetos, y por lo tanto no hay clases.

La placa Arduino Nano utilizada en este proyecto tiene un solo núcleo, con lo que no tiene sentido implementar complejas planificaciones de tiempo real, pero tampoco son necesarias. No se ha modificado el software de la placa, y los algoritmos utilizados se han implementado con el IDE nativo de Arduino.

Se ha utilizado una planificación cíclica, concretamente el muestreo periódico, utilizando la función nativa *loop*() y variables globales para controlar el periodo para cada operación.

La recepción de mensajes por puerto serie se realiza de forma asíncrona mediante la función *serialEvent*() de la arquitectura *Arduino*.

Las funciones que se han implementado, y que serán de uso común para todos los módulos, como el envío y recepción de mensajes *MQTT*, la traducción de identificadores *MQTT* a número de Pin, etc. deberán ser movidas a una o varias clases C++ y utilizadas por los módulos como una librería externa, si finalmente esta solución deja de ser un prototipo para ser un proyecto final.

Raspberry PI

Como ya se ha explicado, la Raspberry PI será la encargada de gestionar los Módulos *Arduino*, y liberarlos de toda la complejidad, hardware y software, que implica la comunicación y sincronización con el resto de la plataforma.

Se ha elegido *Visual Studio C++* por las facilidades que aporta para desarrollar y depurar en esta plataforma.

La arquitectura está dividida funcionalmente en tres partes:

- **Gestión MQTT**: Se ha elegido la librería *Eclipse Paho*, para realizar la comunicación con el servidor *MQTT* de forma asíncrona utilizando el patrón Observador *[Observer01]* .

Clases	Descripción
MsgSubject	Encargada del envío de mensajes MQTT
MqttConnector	Recepción asíncrona de mensajes MQTT
MsgObserver	Observador (Listener) MQTT

TopicList	Clase para la gestión de listas de Topics

- **Gestión de Módulos**: Es la clase principal, la encargada de sincronizar los módulos y la comunicación *MQTT*.
  - Detecta todos los módulos conectados a USB de forma automática, sin necesidad de configuración previa.
  - Se encarga de realizar las subscripciones *MQTT* para los módulos detectados.
  - Se encarga del enrutado de mensajes entre el bus serie y la red *MQTT*

Clases	Descripción
ModuleManager	Clase principal de gestión de módulos

- **Módulos Serie**: Son las clases encargadas de implementar la gestión y comunicación con los módulos Arduino conectados vía USB. La arquitectura está pensada para implementar de forma transparente al *ModuleManager* distintos tipos de módulos, como, por ejemplo, un módulo que gestione la propia *Raspberry*, módulos para comunicaciones CAN, o incluso ethernet.

Clases	Descripción
SubModule	Clase abstracta para módulos
SerialModule	Implementación de módulos serie (USB)

En las siguientes figuras podemos ver el diagrama de las clases mencionadas:

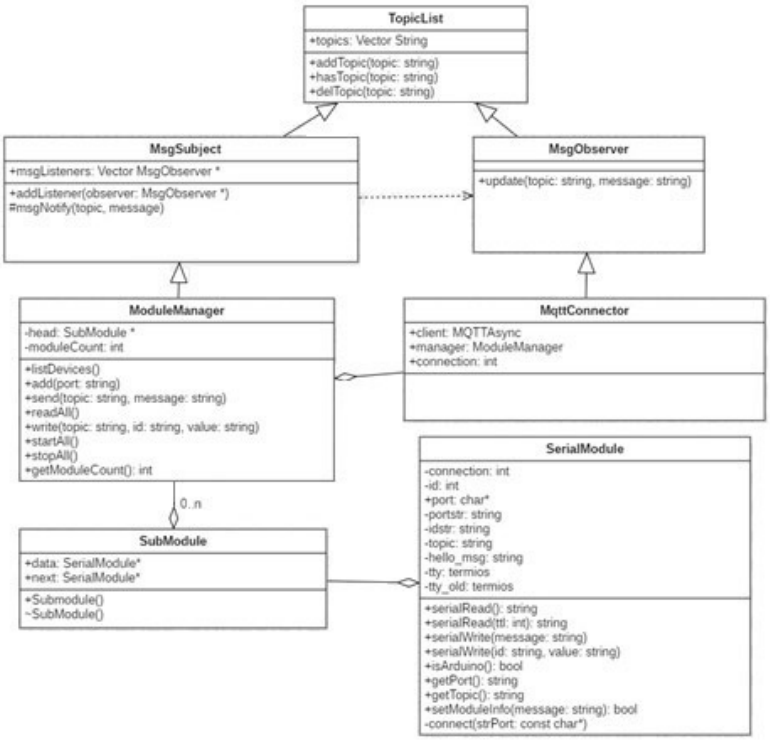


Diagrama de clases, Gestor comunicaciones IoT

Simulador

El simulador es el encargado de inicializar y configurar los diferentes módulos, cargar los ficheros de simulación y lanzar las diferentes secuencias de acciones que forman parte de cada simulación.

Para esta pieza se ha elegido Java para el negocio, y *JavaFx* para la parte visual. Java ofrece unas buenas características multiplataforma, una amplia comunidad y un entorno de desarrollo gratuito.

La arquitectura estará dividida en dos partes bien diferenciadas:

- Lógica de negocio: Se trata de un proyecto que implementará el modelo de la aplicación, y que puede funcionar de forma totalmente autónoma, ejecutándolo en línea de comandos. Las capacidades multimedia de la lógica de negocio son más reducidas, acordes con una ejecución en línea de comandos.
  Está totalmente implementada en Java 1.8
- Lógica de presentación: Es la parte visual de la aplicación, utilizando el patrón MVC de *JavaFX*, que será la encargada no solo de la parte visual de la aplicación, sino también de ampliar las capacidades multimedia de la lógica de negocio.
  También se encarga de extender las clases de la lógica de negocio para adaptarlas a los eventos y tipos de datos de *JavaFx*.

Lógica de negocio

Se trata de la implementación del modelo de la aplicación, que pretende ser lo más genérica e independiente de la plataforma posible.

La arquitectura está dividida funcionalmente en dos partes:

- **Control de comunicaciones MQTT**: Clases encargadas de la recepción y envío de mensajes al gestor de contexto *MQTT*. Se ha utilizado como base la librería *Eclipse Paho*, al igual que en la Raspberry, pero esta vez en su versión para Java. También se ha implementado el patrón Observador para realizar comunicaciones de forma asíncrona.
- **Ejecución de simulaciones**: Tiene como clase principal *SimulatorRunner*, y se encarga de la carga de ficheros de simulación y de su ejecución.

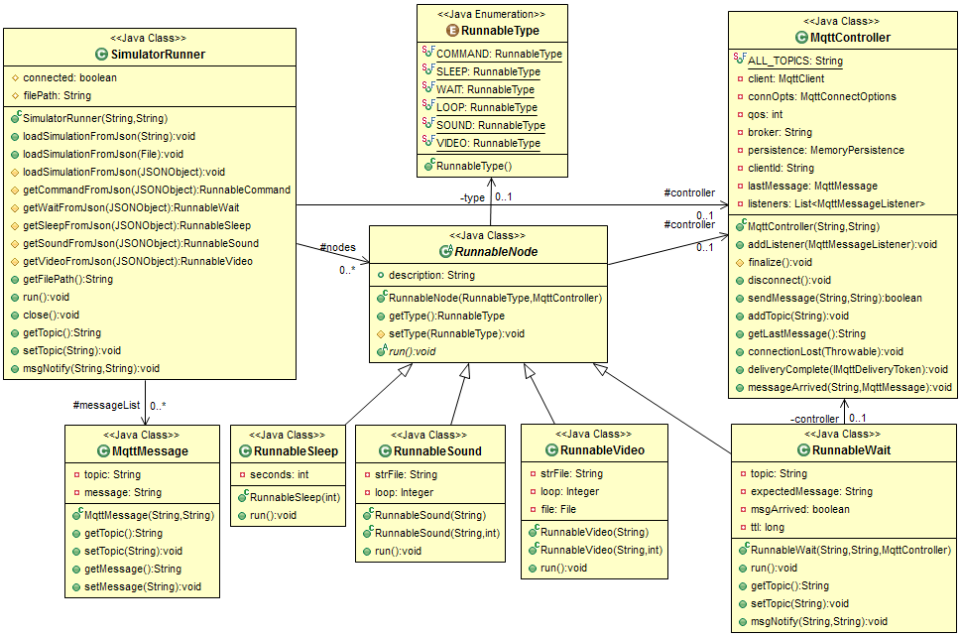


Diagrama de clases, lógica de negocio

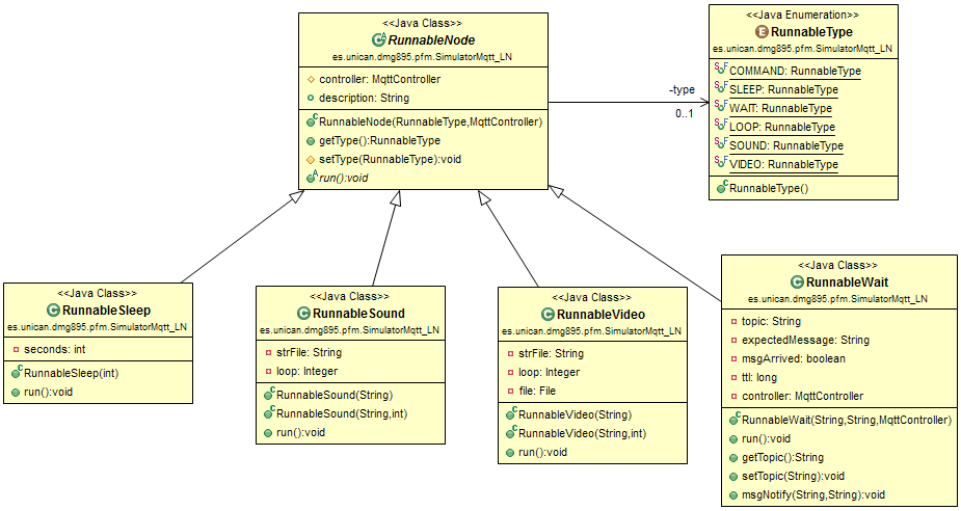


Diagrama de clases, RunnableNode

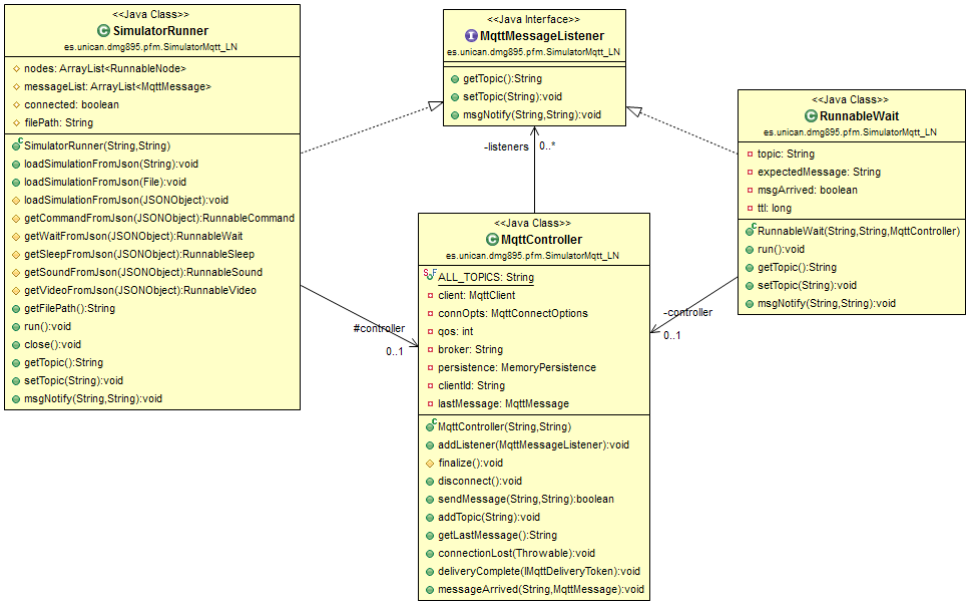


Diagrama de clases, SimulatorRunner

Lógica de Presentación

Es la encargada de la interfaz de usuario con *JavaFx*. Sigue el patrón MVC extendiendo las clases de la lógica de negocio para adaptarlas a las necesidades de la interfaz gráfica y aprovechar todas las capacidades que nos ofrece *JavaFX*.

Además de extender y adaptar el negocio para *JavaFx* se ha hecho necesaria la implementación de nueva funcionalidad, directamente asociada al motor elegido para la interfaz gráfica.

- *Modelo JavaFx*

Comprende las clases necesarias para adaptar el negocio a las capacidades y requisitos específicos de *JavaFx*.

Extensión del modelo para JavaFx:

Las principales funcionalidades que se consiguen con esta extensión del negocio son:

- Utilización de tipos de datos *JavaFX* "observables" y que permiten vinculaciones directas con elementos visuales.
- Añadida funcionalidad que permite mostrar el estado de la ejecución.
- Implementación de funcionalidad de representación geográfica (mapa).
- Funcionalidad *MQTT* propia del simulador, notificaciones y reporte de estado de simulación por *MQTT*.
- División de la funcionalidad en distintos hilos de ejecución para soportar una interfaz de usuario multitarea.

Clases	Descripción
SimulatorRunnerFx	Extensión del negocio para *JavaFX*
SimulationNode	Extensión de RunnableNode del negocio.
SimulationEntity	Entidades representables en el mapa.

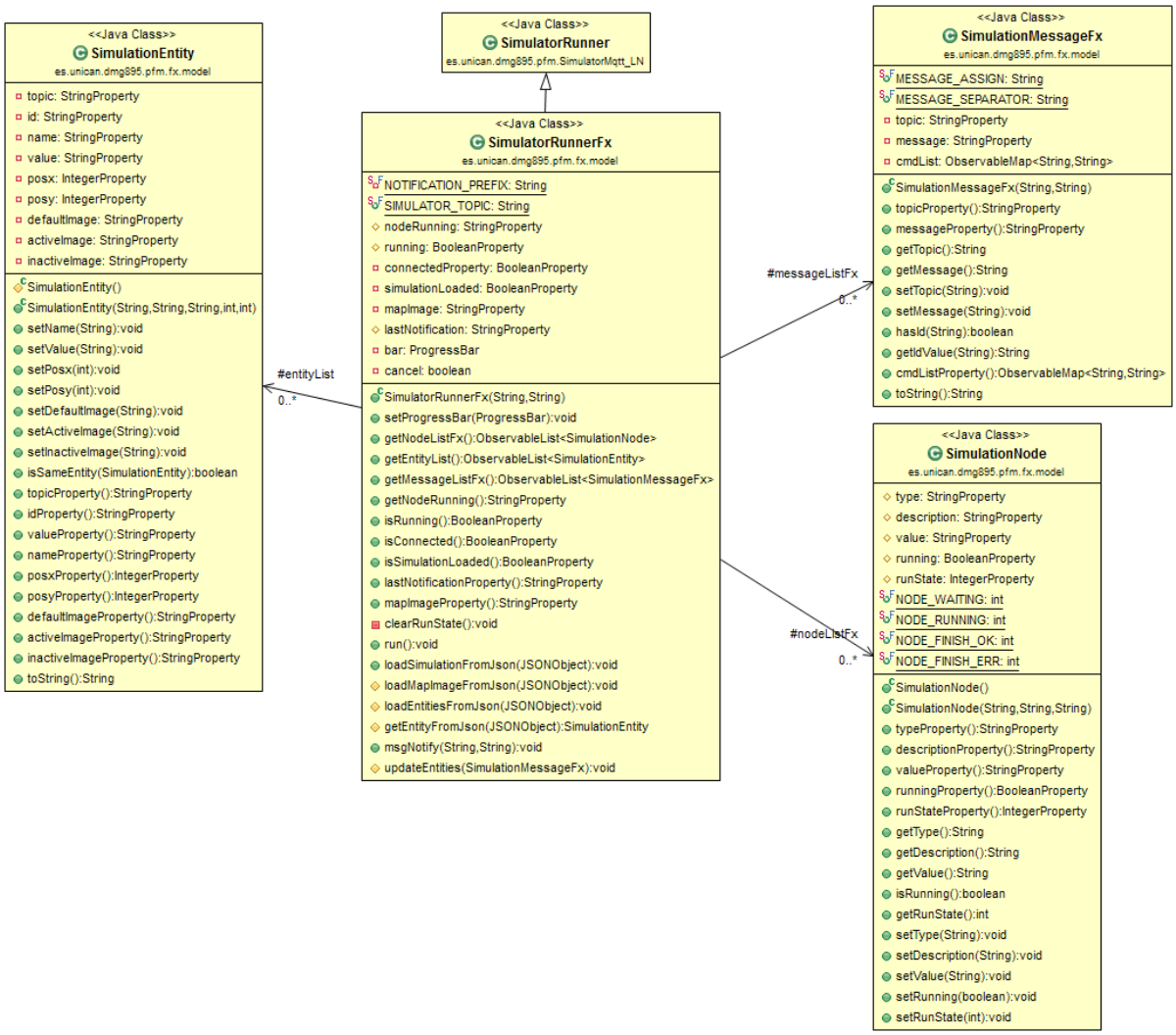


Diagrama de clases, lógica de negocio JavaFx

- *Controlador JavaFx*

Se trata de los controladores asociados a las diferentes vistas (pantallas o paneles) de la presentación. Es la encargada de la funcionalidad de la parte visual, el control de eventos y de "conectar" la parte visual (vista) con el negocio de la aplicación (modelo).

En este caso tendremos dos: el controlador encargado de la gestión de la ventana principal de la aplicación y un controlador específico para controlar tanto el panel del mapa como el de notificaciones.

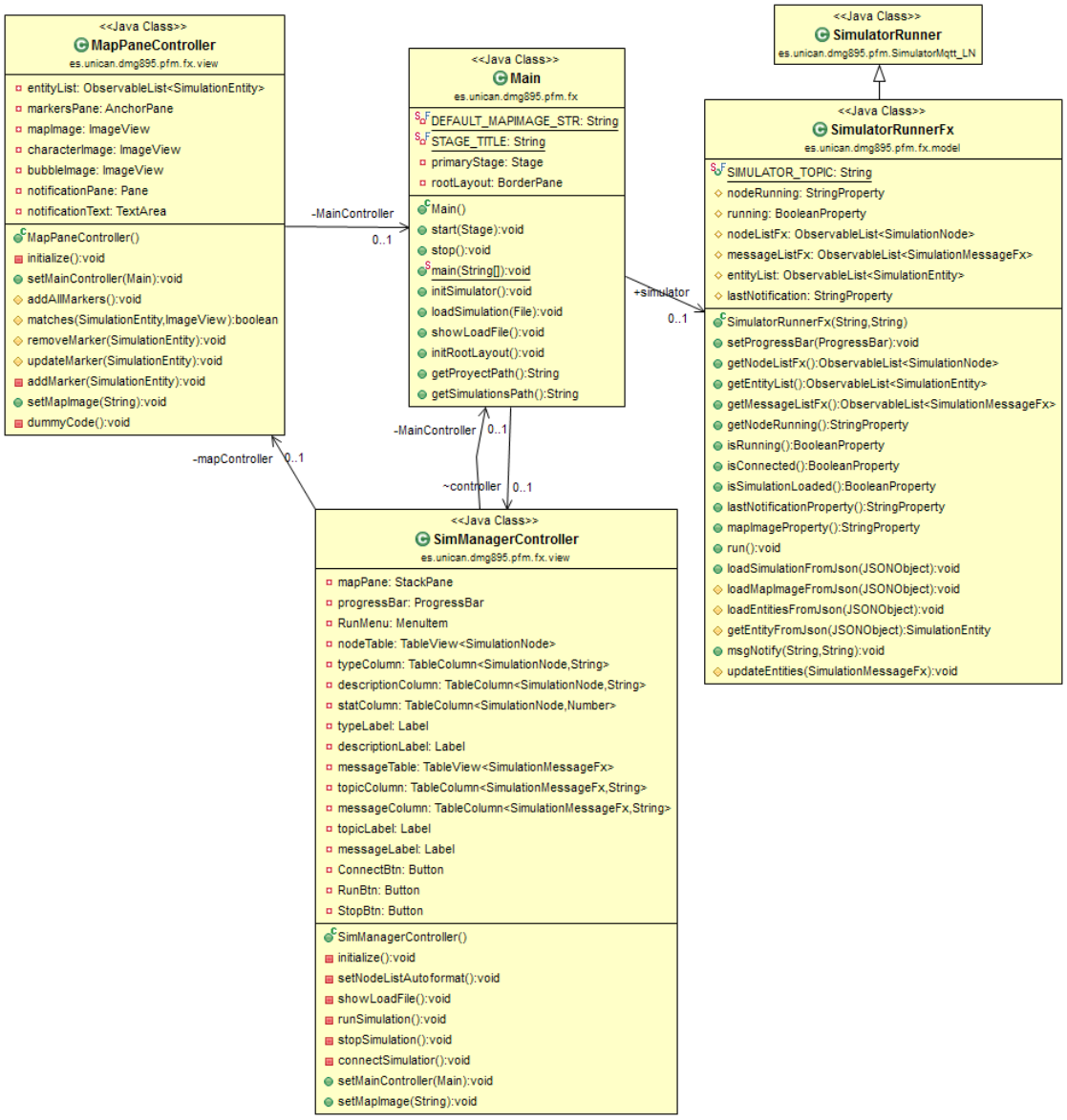


Diagrama de clases, Controlador JavaFx

Control JavaFx:

Clases	Descripción
Main	Controlador principal de la aplicación
SimManagerController	Controlador de la Vista *(página)* principal.
MapPaneController	Controlador del Mapa

- *Vista JavaFx*

Es la capa encargada de la definición de los componentes visuales de la interfaz de usuario. Define el tipo de controles que se van a mostrar, su estética, etc.

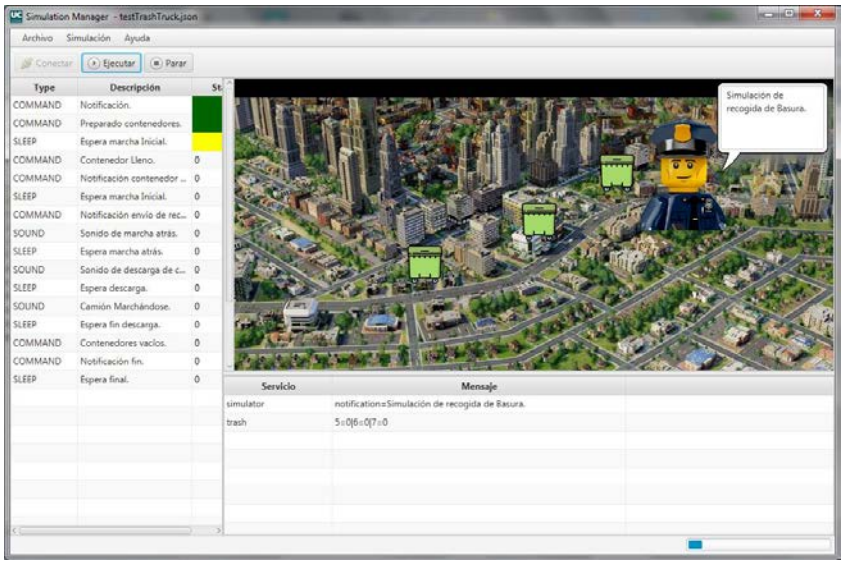
La vista está implementada mediante varios ficheros con extensión *fxml*:

- SimManager.fxml: Contiene la definición de la interfaz de usuario de la ventana principal, con las barras de herramientas y las ventanas de simulación y consola.

- MapPane.fxml: Contiene la definición de la interfaz de usuario del panel que contiene el mapa, y que se muestra dentro de la ventana principal. También contiene la gestión de marcadores y los diálogos emergentes (popups) de notificación.

3.2 Diseño de la Interfaz

3.2.1 Gestor de simulaciones



Gestor de Simulaciones

Como ya se ha explicado, el software de gestión de simulaciones es el encargado de lanzar las secuencias de simulación y de reproducir el contenido multimedia asociado a las mismas. Se trata de una aplicación construida en *JavaFx* sobre un proyecto que implementa la lógica de negocio. Toda la arquitectura está pensada para realizar comunicaciones asíncronas, utilizando múltiples hilos de ejecución.

Para cumplir esta función se ha dividido la interfaz gráfica en varias partes claramente identificables:

- **Barra de menús y herramientas:** Típica barra de herramientas con los controles para cargar, ejecutar y parar simulaciones.
- **Tabla de secuencia de simulación:** Situada en la parte izquierda de la interfaz, muestra la secuencia de acciones de la simulación cargada, así como una descripción y su estado de ejecución.
- **Mapa:** Se trata de una representación gráfica del estado de la simulación. Es capaz de representar marcadores personalizados, así como su estado. A destacar también que el mapa contiene un **área de notificación,** también personalizable y que muestra comentarios o información programada como parte de la simulación.
- **Consola MQTT:** En la consola podremos ver todo el intercambio de mensajes *MQTT* que se están produciendo en la plataforma durante la ejecución de la simulación.

Ficheros de Simulación

Los ficheros de simulación están compuestos por un fichero JSON que especifica la secuencia y parámetros de simulación, y todos los ficheros auxiliares, como imágenes, sonidos o vídeo.

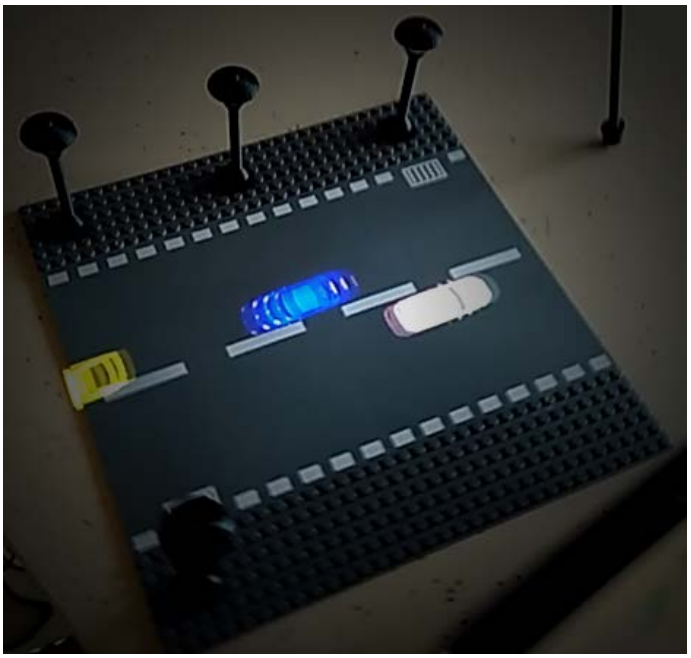
La estructura del fichero es la siguiente:

- **mapimage**: *(Opcional)* Fichero de imagen utilizado como mapa en el simulador. Si la ruta es relativa se calcula a partir de la ubicación del fichero de simulación.
- **nodes**: Array con la secuencia de nodos que formarán parte de la simulación. Puede ser de los siguientes tipos:
  - **command:** Envía un comando MQTT al servidor.
    - **type:** Tipo de nodo, en este caso será ***command***.
    - **topic:** Topic MQTT al que se enviará el mensaje.
    - **value:** Mensaje MQTT a enviar.
    - **description:** Descripción del nodo, se mostrará en el gestor de simulaciones.

    Los comandos enviados al *topic*: **simulador**, con el id de entidad **notification**, se mostrarán en el área de notificación del mapa, en la parte superior derecha.
  - **sleep**: Pausa la simulación el tiempo especificado.
    - **type:** Tipo de nodo, en este caso será ***sleep***.
    - **value:** Tiempo de espera en segundos.
    - **description:** Descripción del nodo, se mostrará en el gestor de simulaciones.
  - **wait**: Pausa la simulación esperando una señal MQTT
    - **type:** Tipo de nodo, en este caso será ***wait***.
    - **topic:** Topic MQTT del que se espera el mensaje.
    - **message:** Mensaje que se espera recibir.
    - **description:** Descripción del nodo, que se mostrará en el gestor de simulaciones.
  - **sound**: Ejecuta un sonido de forma asíncrona, esto es, lanza el sonido y continua la simulación sin esperar a que termine.
    - **type:** Tipo de nodo, en este caso será ***sound***.
    - **value:** Ruta del fichero de sonido a reproducir.
    - **description:** Descripción del nodo, que se mostrará en el gestor de simulaciones.
  - **video**: Reproduce un sonido de forma asíncrona, esto es, lanza el vídeo sin esperar a que termine.
    - **type:** Tipo de nodo, en este caso será ***video***.
    - **value:** Ruta del fichero de video a reproducir.
    - **description:** Descripción del nodo, que se mostrará en el gestor de simulaciones.
- **entities**: *(Opcional)* Lista de entidades para el simulador. Por el momento se trata de marcadores para el mapa, si bien en un futuro sería interesante implementar más tipos, como formas, líneas, animaciones, etc.
  - **topic:** Topic MQTT al que está asociada la entidad.
  - **id:** Id MQTT al que está asociada la entidad.
  - **value:** Valor que se corresponde con el ON.
  - **defaultimage:** Imagen para mostrar mientas no se tengan "noticias" de la entidad.
  - **onimage:** Imagen del marcador cuando llegue el valor esperado.
  - **offImage:** Imagen del marcador para el resto de los valores.
  - **posx:** Posición X en la que se mostrará el marcador sobre el mapa, en pixeles.
  - **posy:** Posición Y en la que se mostrará el marcador sobre el mapa, en pixeles.
  - **virtual:** Indica si se trata de una entidad real o virtual

3.2.2 Proyección multimedia

Todas las simulaciones pueden reproducir vídeos, que pueden mostrarse en un monitor o una pantalla de proyección. Pero el objetivo principal que se buscaba cuando se pensó en esta funcionalidad es la proyección directa sobre la maqueta, esto nos abre una infinidad de posibilidades para enriquecer las simulaciones, proyección de tráfico, cambios meteorológicos, proyección sobre fachadas de edificios, etc.



Primer prototipo de proyección de tráfico.

3.2.3 Módulo de simulación de alumbrado

Se trata de un pequeño módulo, controlado por un *Arduino Nano* y encargado de demostrar capacidades de simulación para un servicio municipal de alumbrado público.

Módulo de simulación de alumbrado

Las funcionalidades que se han implementado son:

- **Telecontrol individual de luminarias:** Se puede controlar el encendido y apagado de las luminarias de forma individualizada mediante el envío de mensajes *MQTT* a la plataforma.
- **Control automático de luminarias:** El módulo de simulación contiene un sensor de luminosidad que controla de forma automática el encendido de las luminarias en función de la luz ambiental. Esta funcionalidad se puede activar o desactivar de forma remota mediante mensajes *MQTT*. El umbral de luminosidad para activar las luminarias también puede ser modificado de forma remota.
- **Simulación de fallo de luminaria:** Las luminarias pueden hacer una simulación visual de haberse averiado, esta simulación consiste en un parpadeo errático que finaliza en el apagado de la luminaria. Este fallo se puede inducir de forma remota, a través de otro módulo, o por acción de un actuador del propio módulo de luminarias.

El módulo está pensado para controlar hasta 20 luminarias, un sensor de luminosidad, y dos finales de carrera para la simulación de colisiones con las luminarias. Esta limitación se ha hecho teniendo en cuenta las limitaciones de consumo del controlador, ya que en esta versión las luminarias están alimentadas directamente por el Arduino.

3.2.4 Módulo de simulación de Basuras

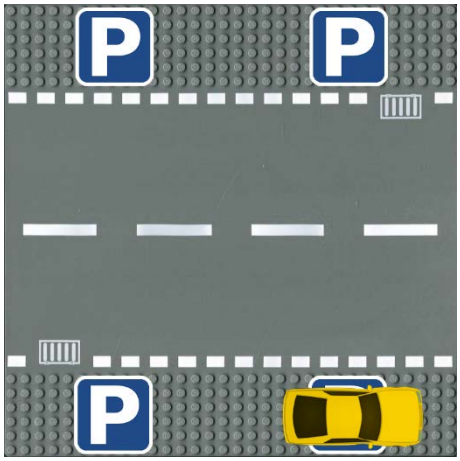
Pequeño módulo controlado por Arduino Nano, encargado de demostrar las posibilidades de simulación de un servicio municipal de recogida de residuos.

Módulo de simulación de basuras

Las funcionalidades que se han implementado son:

- **Detección de llenado:** Con un sensor de infrarrojos, y un pequeño circuito de control diseñado para este proyecto, se ha implementado en una placa de prototipos un sensor binario (lleno/vacío) para detectar objetos dentro del contenedor.
- **Vaciado automático:** Un pequeño servomotor y una compuerta en la parte inferior, se encargan de vaciar el contenedor, bien de forma remota o bien directamente desde el Arduino.

3.2.5 Módulo de sensores de Aparcamiento.



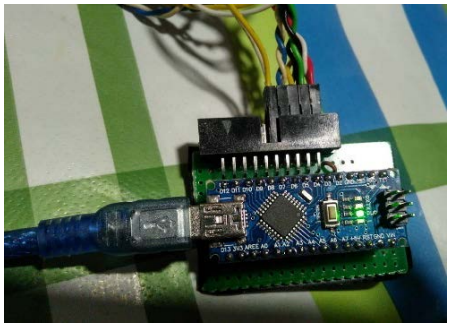
Módulo, también controlado mediante *Arduino Nano*, encargado de simular sensores de aparcamiento.
La funcionalidad que se ha implementado es:

- **Sensor de Parking**: Mediante sensores magnéticos (*Reel*), se detectan las plazas ocupadas, enviando los cambios de estado mediante *MQTT*.

Se coloca un pequeño imán en la parte inferior de los vehículos que será detectado por el sensor situado bajo la placa de aparcamiento.
Estos mismos sensores se pueden utilizar para simular choques (accidentes) contra las farolas.

3.2.6 Placas Prototipo para controladores Arduino.

Para cada controlador *Arduino* se ha diseñado una pequeña placa de prototipos que contiene los conectores necesarios tanto para el controlador, como para los sensores y actuadores, así como circuitería básica para el correcto funcionamiento de los sensores.



Construcción de placa de prototipo

Como es lógico el diseño de la placa de prototipos varía para cubrir las necesidades de cada módulo de simulación.

4. Implementación del Sistema

4.1 Estándares y Normas Seguidos

- **MQTT**: Para toda la comunicación interna de la plataforma.
- **FIWARE:** Para la persistencia en la plataforma real, en la Transformación de *Pentaho Data Integration*.

4.2 Lenguajes de Programación

Como ya se ha comentado, los lenguajes utilizados para la implementación de esta solución son:

- **Arduino (C++):** Para codificar los módulos *Arduino Nano*.
- **C++:** Para el gestor de módulos y comunicaciones de la *Raspberry*.
- **Java:** Para la lógica de negocio del simulador.
- **JavaFX:** Para la interfaz gráfica de usuario del simulador.

4.3 Herramientas y Programas Usados para el Desarrollo

Las herramientas utilizadas han sido:

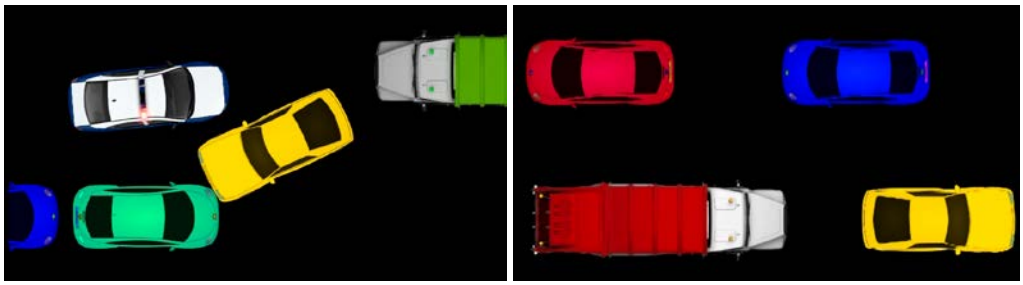
- **Git:** Se ha utilizado Git para realizar el control de versiones de todas las piezas de código, así como para la gestión de tareas, incidencias e hitos.
- **Arduino IDE:** Para codificar los módulos *Nano*.
- **Visual Studio 2017** Community Edition: Para la codificación y compilación cruzada de las Raspberry.
- **Eclipse Mars**: Para la codificación del Simulador.
- **Pentaho Data Integration (Kettle)**: Como parte de las soluciones de inteligencia de negocio (Business Inteligence) que ofrece *Pentaho*, encontramos esta potente herramienta de integración de datos, gratuita y de código abierto, que nos permite enviar datos y estadísticas de la maqueta tanto a un gestor de contexto *FIWARE* (*Orion*) como a una base de datos, para su posterior análisis.

4.4 Generación de contenido multimedia

Para la generación de contenido multimedia y de proyección sobre la maqueta se han utilizado:

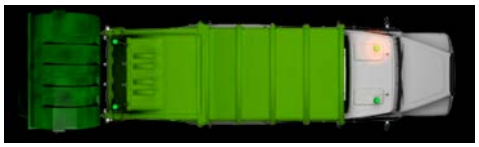
- **Gimp:** Programa gratuito para la edición de imágenes utilizadas en las animaciones.

- **Hitfilm 4 Express:** Programa de edición de vídeo gratuito con el que se han creado las animaciones de tráfico sobre la maqueta.



Fotogramas de la animación de tráfico

- **Blender:** Herramienta de edición y animación 3D utilizada para realizar determinadas animaciones como la de la descarga de un contenedor de basura. También es gratuito.



Capturas de la simulación de residuos

4.5 Módulos de Simulación

Como se ha comentado, se han construido 2 módulos, que de forma conjunta son capaces de simular tres de los servicios presentes en cualquier ciudad moderna. Estos servicios son: alumbrado, gestión de residuos y tráfico, más concretamente la gestión de aparcamientos.

Cada módulo dispone de su propio controlador de Arduino, de tal forma que el primero controla el servicio de alumbrado, y el segundo controlador es compartido para el servicio de gestión de residuos y el de aparcamientos.

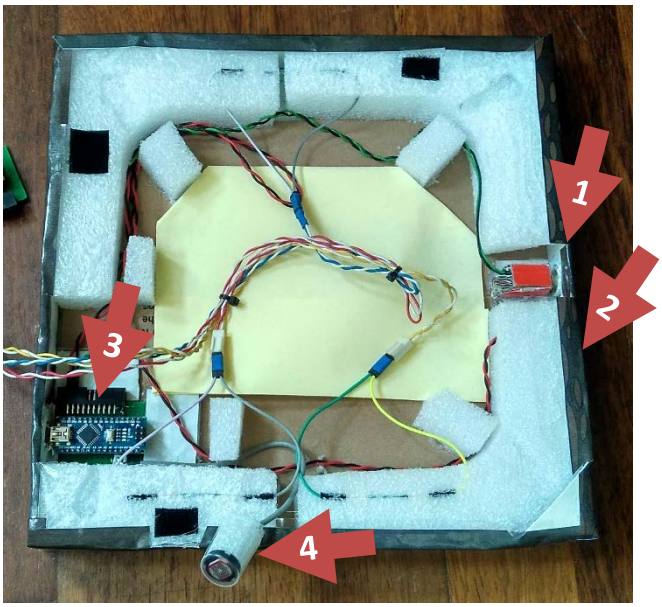
Módulos del prototipo

4.5.1 Módulo de Alumbrado

Está montado sobre una pieza de carretera de Lego y las luminarias se han realizado mediante impresión 3D montando posteriormente en su interior tanto el diodo led como la resistencia necesaria para la conexión directa al controlador Arduino. También dispone de un sensor de luminosidad para permitir el encendido y apagado automático de las luminarias en función de la iluminación ambiental.

Tiene dos modos de funcionamiento, **automático**, en el que se controlan las luminarias en función de la iluminación ambiental, y un modo **pasivo**, en el que el módulo se limita a responder a comandos MQTT.

El cambio de modo de funcionamiento se puede realizar mediante un comando MQTT o bien presionando el pulsador frontal que también dispone de un indicador led que se enciende cuando el módulo esté funcionando en modo automático.



Interior del módulo de Alumbrado

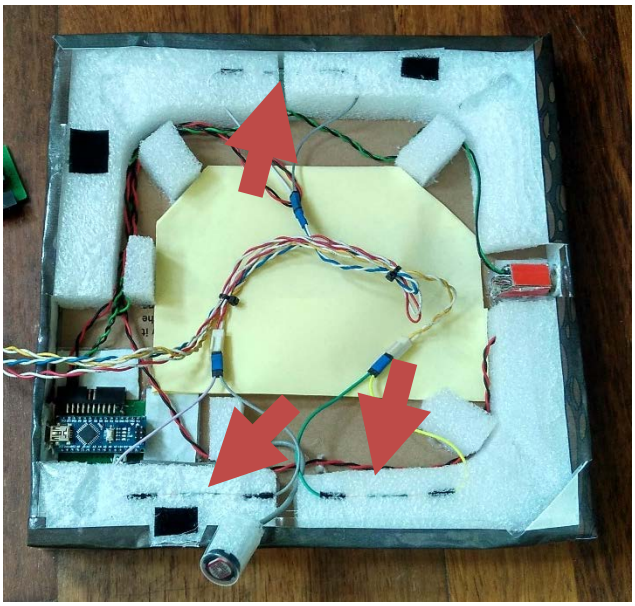
En la imagen podemos ver el interior del módulo resaltando sus partes principales:

1. Pulsador frontal para cambiar entre modo pasivo y automático.
2. Indicador Led de modo automático.
3. Controlador Arduino del módulo.
4. Sensor de Luminosidad.

4.5.2 Gestión de aparcamientos

Como ya se ha comentado, la gestión de aparcamientos se controla desde el controlador de residuos, sin embargo, los sensores magnéticos de aparcamiento están montados en el módulo de alumbrado, que es el que tiene la carretera. No se han conectado estos sensores al sensor de

alumbrado ya que este necesitará todas sus entradas y salidas para las luminarias, mientras que el de gestión de basura tenía todas disponibles excepto dos.



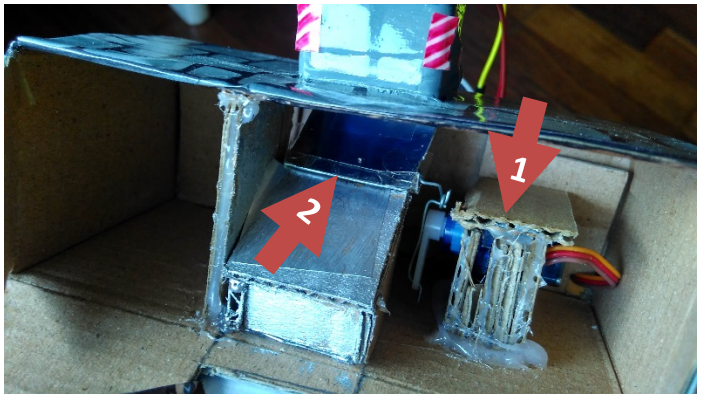
Sensores de aparcamiento

En la imagen se muestra la disposición de los tres sensores de aparcamiento disponibles, que se corresponden con los espacios libres entre las luminarias. También se dispone de un pequeño coche de juguete equipado con un imán para realizar las demostraciones.

Coche "aparcado"

4.5.3 Módulo de residuos

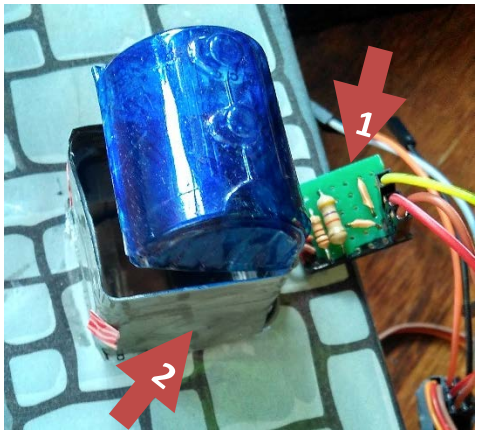
El módulo de residuos está compuesto por un pequeño contenedor de basura. Se ha construido un pequeño circuito para implementar una barrera de infrarrojos que detecte objetos dentro del contenedor, y se ha colocado un servo-motor en el interior del módulo que abre una compuerta en la parte inferior del contenedor para su vaciado. Este motor nos permite validar que la plataforma es capaz de controlar actuadores.



Detalle interior del contenedor

Resaltado en la imagen del interior del módulo podemos ver:

1. Servo-motor.
2. Compuerta de vaciado.



Detalle exterior del contenedor

En el detalle exterior vemos:

1. Circuito de control de barrera de infrarrojos.
2. Sensor de infrarrojos, y en el lado opuesto un led completa la barrera.

4.6 Vídeos de demostración

Junto con esta documentación se adjuntan videos demostrativos del prototipo funcionando para poder justificar los resultados obtenidos.

5. Conclusiones

5.1 Conclusiones

Se ha diseñado y construido un prototipo de maqueta totalmente funcional, compuesto por módulos capaces de funcionar de forma tanto conjunta como independiente, escalable y con un coste muy reducido.

Todos estos resultados justifican que el proyecto no solo es viable, sino que con una complejidad razonable es capaz de aportar unos resultados muy llamativos por encima de las expectativas iniciales.

El diseño permite añadir nuevos módulos y simulaciones de forma transparente, plug & play, que permitirán a la maqueta adaptarse a los cambios constantes que, a buen seguro, se harán en el discurso que se transmitirá al ciudadano.

Será posible utilizar las diferentes piezas de la maqueta de forma conjunta, o de forma individual, es decir, se puede utilizar los módulos *Raspberry* sin simulador, y el simulador también puede ser utilizado sin una maqueta física, o incluso sin la interfaz gráfica de usuario.

En definitiva, la arquitectura escogida es viable dentro de los parámetros que se formularon con los requisitos iniciales.

La obtención de un primer producto final requerirá de pequeños ajustes de software, mas orientados a la usabilidad que a la funcionalidad, lo que avala la decisión inicial de realizar este prototipo como el método mas adecuado para abordar el proyecto.

5.2 Ampliaciones

Como es lógico en cualquier prototipo, quedan en el tintero multitud de ideas y mejoras que podrían ser implementadas en un futuro.

5.2.1 IoT

- Comunicación entre módulos.
- Nuevos tipos de comunicación para módulos, ej. Wifi.
- Aumentar el amperaje que son capaces de controlar los módulos.
- Implementar notificación física de estado, por ejemplo, leds.

5.2.2 Control comunicaciones IoT

- Implementar submódulo de control de la entradas y salidas de la propia *Raspberry*.
- Implementar submódulos de control mediante otro tipo de comunicación.

- Gestión y notificación de submódulos vía *MQTT*.
- Mejorar la parametrización de la configuración.
- Añadir interfaz web de gestión y configuración.

5.2.3 Simulador

- Implementar App de visualización del mapa independiente, para mostrar cara al ciudadano durante las demostraciones, y que esté sincronizada con el simulador vía *MQTT*.
- Implementación de nuevos estados de simulación, como condiciones y bucles.
- Implementar reproducción de vídeo a través de *JavaFx*; ahora se utiliza la reproducción automática del S.O.
- Controles de simulación. (Siguiente, Atrás, etc.)
- Personalizar imagen de notificaciones en el mapa en las propiedades JSON.
- Añadir funcionalidad a los marcadores del mapa, multi-estado, switch por valores, etc.
- Añadir nuevos tipos de entidad al mapa. Polígonos, líneas, rutas, etc.
- Implementar módulos de visión por computador y/o realidad aumentada.

6. Glosario

A

Arduino
Mini controlador de bajo coste para desarrollo de proyectos electrónicos3, 4, 13, 15, 16, 17, 18, 19, 28, 29, 31, 32, 40, 41, 42, 43

C

C++
Lenguaje de programación orientado a objetos..3, 4, 5, 17, 18, 19, 25, 31, 43

Context Broker
Gestor de contexto, pieza software encargada de registrar y distribuir el último valor conocido de un conjunto de datos 3, 5, 17, 18, 30

E

ETL
Proceso de extracción transformación y carga de datos. (Extract, Transform, Load).....18, 29

F

FIWARE
Conjunto de componentes de código abierto para acelerar el desarrollo de solucinoes SMART9, 12, 13, 17, 43

I

IoT
Internet de las cosas (Internet Of Things)3, 4, 5, 13, 14, 15, 16, 27, 28, 32, 33, 44, 47

J

JSON
JavaScript Object Notation, Formato de intercambio de datos 22, 39, 45

JUnit
Librerías para desarrollo de pruebaas unitarias en Java.....25

K

Kettle
Nombre informal de la herramienta Pentaho Data Integration (PDI).....29, 43

kpi
Key Performance Indicator o Indicador de rendimiento 17

M

MQTT.
Protocolo de mensajería del tipo publicación-subscripción pensado para su utilización en el Internet de las cosas13, 19, 29, 31, 34, 35, 41, 42, 44, 45

MVC
Patrón de desarrollo de software, Modelo - Vista - Controlador 30, 33, 35

N

NGSI
Protocolo que especifica todo el ciclo de vida de datos contextuales (Inserciones, actualizaciones, subscripcines ...).....12, 17, 18, 47

O

Orion
Gestor de contexto de la plataforma FIWARE...17, 18, 30, 43

P

PAHO
Librería encargada de la gestión de comunicaciones MQTT.....18, 19

Pentaho BI
Conjunto de aplicacines pensadas para generar Inteligencia de negocio (Business Inteligence) a partir de gran volumen de datos.....29

Pentaho Data Integration
Herramienta de transformación y carga de datos de la plataforma Pentaho BI.....18, 29, 30, 43

R

Raspberry
Mini ordenador compacto.3, 5, 13, 16, 17, 18, 19, 25, 27, 28, 29, 31, 32, 34, 43, 44

7. Referencias Bibliográficas

7.1 Libros y Artículos

REFERENCIAS

[DelRivero2017] Del Rivero Marieta, "Smart Cities, una visión para el ciudadano", LID Editorial Empresarial, 2017, ISBN: 9788416624133

[Telefónica2016] Ontiveros Emilio; Vizcaíno Diego; López Sabater Verónica, "Ciudades del futuro: inteligentes, digitales y sostenibles", Ariel S.A., 2016, ISBN: 978-84-08-17024-2

7.2 Referencias en Internet

REFERENCIAS

[SmartSantander] Primera fase del proyecto Santander Smart City. http://www.smartsantander.eu/index.php/testbeds/item/132-santander-summary Accedido en 25/06/2019

[Fiware01] Iniciativa Open Source para definir estándares universales de gestión de contexto IoT. https://www.fiware.org/ Accedido en 25/06/2019

[Fiware02] Especificación del estándar del protocolo de comunicación NGSI. http://fiware.github.io/specifications/ngsiv2/stable/ Accedido en 25/06/2019

[Paho01] Cliente MQTT Open Source Definido por eclipse. https://www.eclipse.org/paho Accedido en 25/06/2019

[MQTT01] Protocolo de comunicación Machine to Machine M2M. http://mqtt.org/ Accedido en 25/06/2019

[JFX01] Comunidad de desarrolladores JavaFx. https://openjfx.io/ Accedido en 25/06/2019

[Technopedia01] Enciclopedia virtual sobre tecnologías de la información. https://www.techopedia.com/definition/31494/smart-city Accedido en 25/06/2019

[JSON 01] Javascript Object Notation https://www.json.org/ Accedido en 25/06/2019

[MVC01] Patrón Modelo-Vista-Controlador, IBM, https://www.ibm.com/support/knowledgecenter/es/SSZLC2_8.0.0/com.ibm.commerce.developer.doc/concepts/csdmvcdespat.htm Accedido en 25/06/2019

[Observer01] Oscar Belmonte Fernández, "Patrón Observador" , http://www3.uji.es/~belfern/Docencia/Presentaciones/ProgramacionAvanzada/Tema2/observador.html Universidad Jaume I. Accedido en 25/06/2019