Universidad
Politécnica
de Cartagena

industriales
etsii UPCT

CONTRIBUCIÓN A LA INFRAESTRUCTURA FI-WARE. INTEGRACIÓN DE DISPOSITIVOS EMPOTRADOS DE BAJO COSTE EN EL ECOSISTEMA

Titulación: Ingeniero Industrial
Alumno/a: Enrique Hernández Zurita
Director/a/s: Juan Antonio López Riquelme
Andrés Iborra García

Cartagena, 23 de Septiembre de 2014

Agradecimientos

QUIERO DAR MI MÁS SINCERO AGRADECIMIENTO A JUAN ANTONIO LÓPEZ RIQUELME Y ANDRÉS IBORRA GARCÍA POR HABERME GUIADO EN ESTE PROYECTO DE FINAL DE CARRERA. POR TODAS LAS ATENCIONES, POR EL TIEMPO QUE HAN INVERTIDO CONMIGO, Y SOBRE TODO POR SU APOYO Y PACIENCIA. SIN ELLOS NADA DE ESTO HUBIERA SIDO POSIBLE.

TAMBIÉN ME GUSTARÍA DESTACAR LA PACIENCIA QUE HA TENIDO PARA CONMIGO TODO EL EQUIPO DE FI-WARE, ESPECIALMENTE A FERMÍN GALÁN MÁRQUEZ, SUFRIDOR CONSTANTE DE PREGUNTAS PARA LA RESOLUCIÓN DE LOS DISTINTOS PROBLEMAS SURGIDOS CON EL ESTUDIO DE LA PLATAFORMA.

POR ÚLTIMO NO ME QUIERO OLVIDAR DE MIS COMPAÑEROS DE FACULTAD, FAMILIA Y AMIGOS, QUE GRACIAS A ELLOS TODO ES SIEMPRE MUCHO MÁS FÁCIL.

GRACIAS A TODOS.

Índice

Capítulo 1

Introducción

1.1 Introducción.

Hoy en día la sociedad avanza hacia un mundo globalizado en el que cada vez se hacen menos importantes las distancias que separan unos países de otros y donde se hace cada vez más sencillo la interacción entre ellos. Todo ello se hace posible gracias a Internet y a su constante crecimiento y desarrollo.

En los últimos años, se han venido desarrollando tecnologías que están consiguiendo que se minimice el hardware empleado por los usuarios, se consiga una mayor accesibilidad a los datos y que mejore la sociedad mediante la creación de servicios para las ciudades y el hogar. Estas tecnologías son el *Internet de las Cosas* o *Internet of Things (IoT)* y Ciudades Inteligentes *o Smart Cities*.

El crecimiento vertiginoso del número de dispositivos conectados a la red cada año hace patente que se está experimentando un cambio, uno que hará que en el año 2020 se encuentren en la red más de 40.000 millones de dispositivos. La conexión de estos dispositivos hace más cómoda la vida de los usuarios, ya que posibilita acceder a ellos desde cualquier lugar, actuar sobre ellos y comprobar su estado en cualquier momento desde cualquier sistema tipo PC, Tablet o Smartphone. La gran mayoría de electrodomésticos y componentes electrónicos del hogar tendrán conexión a Internet y software especializado para su utilización y control remoto.

En el ámbito de las Ciudades Inteligentes, esta tecnología permite desarrollar aplicaciones nunca vistas, creando grandes redes de sensores y actuadores que pueden recoger una vasta cantidad de información en tiempo real o creando históricos, que posteriormente son analizados gracias a las aplicaciones basadas en *BigData*. Esto desemboca en sociedades más controladas y con mejoras en la eficiencia, tanto energética como económica. Concretamente, son innumerables los servicios que pueden crearse y desarrollarse en virtud de la mejora de las ciudades, tanto para los ciudadanos como para las autoridades gubernamentales.

Además, los novedosos servicios de computación en la nube o *cloud computing* hacen que todo el análisis, cálculo y manejo de datos se haga también a través de Internet, o como actualmente se denomina, en la nube o *cloud*. Además, no se precisará de hardware propio para todo ello, ya que todo es ejecutado en servidores remotos, de la misma forma que ocurre con el almacenamiento.

Debido a esta creciente tendencia, existen actualmente diversas plataformas que hacen posible la conexión de dispositivos, almacenamiento de datos y la computación en la nube, de modo que puedan desarrollarse aplicaciones y servicios como los ya citados.

1.2 Objetivos.

El presente proyecto tiene como objetivo principal estudiar la plataforma para Internet de las Cosas FI-WARE e implementar un caso práctico de estudio empleando en él dispositivos empotrados de bajo coste. Para conseguir este objetivo, se plantea llevar a cabo los siguientes sub-objetivos:

- Estudio de la plataforma FI-WARE.
- Búsqueda y selección de los dispositivos empotrados.
- Estudio, tanto a nivel hardware como software, de los dispositivos seleccionados.
- Diseño de los componentes hardware y software necesarios para implementar el caso de estudio.
- Diseño y creación del widget/es que es/son necesarios para la visualización web de la información del caso.
- Pruebas y puesta a punto.
- Redacción de la memoria final de memoria de proyecto

1.3 Desarrollo de la memoria.

Capítulo 1: Introducción

El Capítulo 1 presenta la motivación de este trabajo, además de enumerar los objetivos del mismo y describir la memoria presentada.

Capítulo 2. Estado del Arte

En el Capítulo 2 se expone el panorama actual de las nuevas tecnologías de Internet de Las Cosas y Ciudades Inteligentes, se describen los dispositivos empotrados de bajo coste más importantes existentes en el marcado y algunas de las plataformas más reconocidas creadas como ecosistemas para Internet de las Cosas.

Capítulo 3. Descripción del sistema.

En este capítulo se seleccionan los dispositivos empotrados y la plataforma a utilizar de entre las expuestas en el capítulo anterior, argumentando las causas de dicha elección. Posteriormente, se describen todos los elementos utilizados con un mayor nivel de detalle.

Capítulo 4. Caso de estudio. Hardware y software desarrollado.

Se desarrolla un caso de estudio concreto en el que se emplean los elementos seleccionados en el capítulo anterior. Se detallan las características principales, el software creado y el hardware.

Capítulo 5. Conclusiones y trabajos futuros.

En este capítulo se enumeran las principales conclusiones tras la realización de este trabajo, así como los futuros trabajos a desarrollar.

Anexo I. Documentación y herramientas para la utilización de Orion CB y Cosmos.

En el Anexo I se muestra todo lo necesario para poder implementar más adelante un caso de estudio. Se muestra con todo detalle cómo crear una instancia en FI-WARE, operar con ella y cómo se puede conseguir la persistencia de datos en un histórico.

Anexo II. Detalle del software desarrollado para el caso de estudio.

En el capítulo anterior se comentan las partes más importantes del software implementado en los dispositivos. Sin embargo, es en el Anexo II donde se exponen de forma más detallada las funciones y sentencias que hacen posible el funcionamiento del sistema.

Capítulo 2

Estado del Arte

2.1 Introducción

El concepto de la nube o *the cloud* surge en la actualidad como un lugar remoto donde almacenar información y prestar diferentes servicios a los usuarios. Junto con la nube aparece el concepto de computación en la nube, conocido también bajo los términos servicios en la nube, informática en la nube, nube de cómputo o nube de conceptos, del inglés *cloud computing*, que se trata de un paradigma que permite ofrecer servicios de computación a través de Internet. El *cloud computing* permite separar el equipo físico del usuario y parte del software y almacenamiento de forma que queden a disposición de dicho usuario desde un servidor remoto en Internet. En este tipo de arquitectura se pueden identificar las principales ventajas y desventajas, las cuales se muestran a continuación.

VENTAJAS

- Reducción de costos: No hay necesidad de adquirir un hardware y un software determinado, lo que reduce costos operativos en infraestructura, mantenimiento y energía. La nube es más barata que la instalación y mantenimiento de un servidor propio o contratar los servicios de un proveedor.

- Flexibilidad: El servicio de nube se paga de acuerdo a la demanda.

- Movilidad: La información al quedar alojada en la nube pueden ser consultados por el usuario desde cualquier lugar.

- Focalización: *Cloud Computing* permite a las compañías centrarse en su negocio principal, en lugar de hacer una alta inversión tecnológica en sistemas. De este modo, una empresa podría invertir en su infraestructura industrial o física o en capital humano para proseguir sus planes de expansión

- Ecología: Usar la nube reduce la huella de carbono en el medio ambiente al ahorrar recursos y componentes que pasan de estar almacenados en componentes físicos a ser virtuales. Se ahorra también en consumo de energía.

DESVENTAJAS

- Seguridad: Se debe ser muy cuidadoso con el manejo de la información para evitar que los datos sean robados o extraviados en agujeros de seguridad.

- Privacidad: Datos confidenciales y sensibles como planes de mercado, lanzamientos de productos, información personal, datos financieros, pueden quedar en manos de terceros si no se tienen las medidas preventivas.

- Conectividad: La velocidad de acceso a la información y la disponibilidad de las aplicaciones dependen de la velocidad de la conexión a Internet. Sin acceso a Internet no hay *Cloud Computing* y este servicio puede caerse en cualquier momento por diversos factores.

- Una vez conocidas las ventajas del *Cloud Computing* se puede tomar la decisión de usarla, pero se debe antes analizar en el mercado a aquellas empresas que ofrecen este servicio y evaluar las características que ofrecen para luego compararlas y escoger la que más beneficios y garantías ofrezcan. Además, es importante evaluar bien la solidez que dicha empresa tiene en el mercado.

Hablar de la nube se hace cada vez más frecuente y hoy en día se están desarrollando tecnología, plataformas y aplicaciones apoyadas en ella. Dos muy importantes, empleadas en este proyecto, son Internet de las cosas o *Internet of Things (IoT)* y Ciudades inteligentes o *Smart Cities*.

Internet de las cosas es un concepto que se refiere a la interconexión digital de objetos cotidianos con Internet. Se busca que cada vez más objetos estén conectados a la red y que puedan ser monitorizados y actuados desde ella. Se calcula que en pocos años habrá conectados a Internet más de 26 mil millones de dispositivos conectados.

Por otro lado, en las *Smart Cities*, que a su vez parte del Internet de las Cosas, se busca crear un ecosistema más controlado, seguro, eficiente y en definitiva, sostenible, mediante la conexión de múltiples dispositivos que analizan las ciudades y actúan sobren ellas. A partir de la implantación de una red de sensores y actuadores sobre una ciudad se puede tener acceso en tiempo real a información nunca antes analizada, con la cual se pueden crear infinidad de aplicaciones y servicios para mejorar social, energética e incluso económicamente.

Por lo general se necesitan dos elementos básicos para la implementación de Internet de las Cosas:

- Dispositivo que estará conectado a la red y que será el suministrador de información y receptor de las actuaciones que se quieran hacer sobre el sistema. En el caso de *Smart Cities* se utilizan dispositivos de tipo empotrado.

- Plataforma que gestionará la información y que permitirá al usuario recibir la información, almacenarla e interactuar sobre el sistema desde cualquier lugar mediante aplicaciones y servicios en la nube.

2.2 Dispositivos empotrados de hardware y software libre

Un sistema empotrado de hardware y software libre, o también llamado sistema embebido, es un sistema de computación diseñado para realizar una o algunas pocas funciones dedicadas frecuentemente en un sistema de computación en tiempo real. Al contrario de lo que ocurre con los ordenadores de propósito general, que están diseñados para cubrir un amplio rango de necesidades, los sistemas embebidos se diseñan para cubrir necesidades específicas.

Los sistemas empotrados contienen unas características que se describirán a continuación:

- El microprocesador, microcontrolador, DSP, etc. Es decir, la CPU o unidad que aporta capacidad de cómputo al sistema. Generalmente de tipo ARM cuando se trata de microprocesador.
- La comunicación adquiere gran importancia en los sistemas embebidos. Lo normal es que el sistema pueda comunicarse mediante interfaces estándar de cableadas o inalámbricas.
- El subsistema de presentación tipo suele ser una pantalla gráfica, táctil, LCD, alfanumérico, etc.

- Se denominan actuadores a los posibles elementos electrónicos que el sistema se encarga de controlar.
- El módulo de E/S analógicas y digitales suele emplearse para digitalizar señales analógicas procedentes de sensores, activar diodos LED, reconocer el estado abierto cerrado de un conmutador o pulsador, etc.
- El módulo de reloj es el encargado de generar las diferentes señales de reloj a partir de un único oscilador principal. El tipo de oscilador es importante por varios aspectos: por la frecuencia necesaria, por la estabilidad necesaria y por el consumo de corriente requerido.
- El módulo de energía se encarga de generar las diferentes tensiones e intensidades necesarias para alimentar los diferentes circuitos del sistema. Usualmente, se trabaja con un rango de posibles tensiones de entrada que mediante conversores ac/dc o dc/dc, se obtienen las diferentes tensiones necesarias para alimentar los diversos componentes activos del circuito.
- Los conversores ac/dc y dc/dc, otros módulos típicos, filtros, circuitos integrados supervisores de alimentación, etc.

Se va a prestar especial interés en la conexión a Internet de los diferentes dispositivos expuestos dado que es de gran importancia para comunicación con la plataforma de *cloud computing* seleccionada.

2.2.1 Arduino

Arduino (ver Ilustración 2.1) es una plataforma de hardware libre, basada en una placa con un microcontrolador y un entorno de desarrollo, diseñada para facilitar el uso de la electrónica en proyectos multidisciplinares. [1]

El hardware consiste en una placa con un microcontrolador Atmel AVR y puertos de entrada/salida. Los microcontroladores más usados son el Atmega168, Atmega328, Atmega1280, ATmega8 y ATmega32u4, por su sencillez y bajo coste que permiten el desarrollo de múltiples diseños. Por otro lado, el software consiste en un entorno de desarrollo que implementa el lenguaje de programación *Processing/Wiring* y el cargador de arranque que es ejecutado en la placa.

Desde octubre de 2012, Arduino también se usa con microcontroladoras CortexM3 de ARM de 32 bits, que coexistirán con las más limitadas, pero también económicas AVR de 8 bits. ARM y AVR no son plataformas compatibles a nivel binario, pero se pueden programar con el mismo IDE de Arduino y hacerse programas que compilen sin cambios en las dos plataformas. Eso sí, las microcontroladoras CortexM3 usan 3,3 V, a diferencia de la mayoría de las placas con AVR, que generalmente usan 5 V. Arduino se puede utilizar para desarrollar objetos interactivos autónomos, o puede ser conectado a software tal como Adobe Flash, Processing, Max/MSP, Pure Data). Las placas se pueden montar a mano o adquirirse. El entorno de desarrollo integrado libre se puede descargar gratuitamente.

Arduino puede tomar información del entorno a través de sus entradas y controlar luces, motores y otros actuadores. El microcontrolador en la placa Arduino se programa mediante el lenguaje de programación Arduino (basado en *Wiring*) y el entorno de desarrollo Arduino (basado en *Processing*). Los proyectos hechos con Arduino pueden ejecutarse sin necesidad de conectar a un computador.

Ilustración 2.1. Vista del sistema Arduino UNO.

Arduino permite conectar diferentes extensiones o *shields* sobre la placa principal, aportando funcionalidades extra al dispositivo sin perder además las conexiones originales (salvo algunas excepciones).

Algunos de los *shields* de mayor utilidad disponibles son aquellos que permiten la conexión a Internet del dispositivo, ya sea mediante conexión inalámbrica (WiFi Shield) o con cable Ethernet (Ethernet Shield), los cuales se muestran en la Ilustración 2.2.

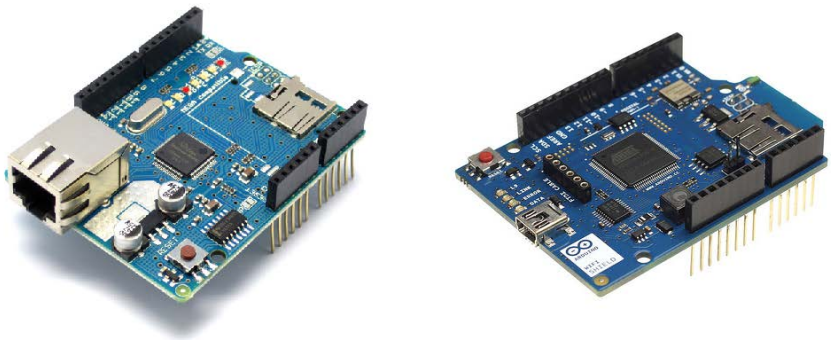

Ilustración 2.2. Vista de los shields disponibles para conexión a Internet.

2.2.2 Pingüino

Pingüino (ver Ilustración 2.3) usa un lenguaje de programación sencillo, basado en Arduino, pero con microcontroladores PIC de Microchip con USB integrado (Una de las principales diferencias con Atmel). Pingüino fue creado a finales del año 2008 por el Ingeniero Electrónico Francés Jean Pierre Mandon en Francia. Jean Pierre observó la posibilidad de diseñar un dispositivo similar a Arduino, pero "portándolo" desde la plataforma de Microcontroladores ATMEGA de ATMEL (de Arduino) a los Microcontroladores PIC de Microchip. [2]

La intención principal de éste cambio fue aprovechar sus conocimientos en la plataforma PIC, y la excelente oportunidad que representaba que algunos modelos de PICs (PIC18F2550 y 4550) integran ya una interfaz USB, que en el caso de Arduino se implementa a través de un chip adicional, que actúa como adaptador/convertidor serial (RS-232) a USB (Universal Serial Bus).

La conexión a Internet se realiza de forma similar a Arduino con controladores de microchip o con módulos Wifi de desarrollo.

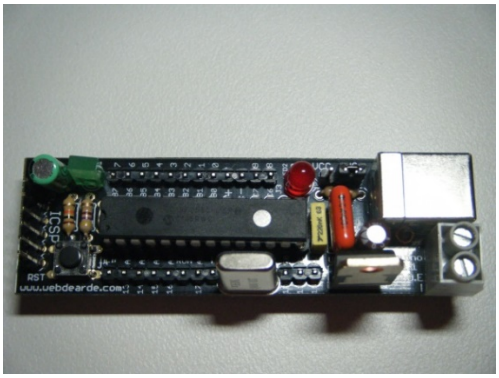

Ilustración 2.3. Vista del sistema pingüino.

2.2.3 Raspberry Pi

Raspberry Pi (ver Ilustración 2.4) es un ordenador de placa reducida o (placa única) (SBC) de bajo costo, desarrollado en Reino Unido por la Fundación Raspberry Pi. El diseño incluye un System-on-a-chip Broadcom BCM2835, que contiene un procesador central (CPU) ARM1176JZF-S a 700 MHz (aunque existe la posibilidad de realizar *overclock* de hasta 1 GHz), un procesador gráfico (GPU) VideoCore IV y 512 MB de memoria RAM. El diseño no incluye un disco duro ni unidad de estado sólido, ya que usa una tarjeta SD para el almacenamiento permanente. Tampoco incluye fuente de alimentación ni carcasa. En febrero de 2012 la fundación empezó a aceptar órdenes de compra del modelo A, en febrero de 2013 del modelo B y enjulio de 2014 el modelo B+. La evolución de un modelo a otro ha sido la adición de puertos USB en cada versión (1 en el modelo A, 2 en el modelo B y 4 en el modelo B+), el incremento de la RAM (de 256 MB en el modelo A hasta 512 MB en el B) y la aparición del puerto Ethernet en la versión B. Además, en la última versión se ha reducido el consumo del sistema.[3]

Dispone de conexiones de audio (Jack 3,5 mm), Ethernet, USB, microUSB para alimentación (5 V), ranura para tarjeta micro-SD, salida HDMI y hasta 40 pines GPIO (*General Purpose Input/Output*) en el último modelo.

En cuanto al software, la fundación da soporte para las descargas de las distribuciones para arquitectura ARM, *Raspbian* (derivada de *Debian*), RISC OS 5, *Arch Linux ARM*

(derivado de *Arch Linux*) y *Pidora* (derivado de *Fedora*), y promueve principalmente el aprendizaje del lenguaje de programación *Python* y otros lenguajes como *Tiny BASIC, C* y *Perl*. Sin embargo, es posible instalar muchos otros sistemas operativos e incluso se ha llegado a implementar Android.

La conexión a Internet se puede realizar mediante una conexión Ethernet, a través del puerto incluido en la placa (a partir del modelo B), o mediante WiFi, conectando un *dongle* USB WiFi.

Ilustración 2.4. Vista Raspberry Pi modelo B+.

2.2.4 BeagleBone

BeagleBone (ver Ilustración 2.5) es un ordenador de placa reducida o *credit-card-sized* basada en el kernel 3.8 de Linux. Es un tipo de *BeagleBoard* creado por Texas Instruments con una filosofía open-source y de bajo coste. Apareció por primera vez en octubre de 2011 y posteriormente, en 2013, apareció un modelo actualizado llamado *BeagleBone Black*. [4]

Implementa un microprocesador ARM Cortex-A8 a 720 MHz ó 1 GHz (dependiendo del modelo), hasta 512 MB DDR3 de memoria RAM, acelerador gráfico 3D y memoria flash eMMC de 2 GB.

En cuanto a conectividad, dispone de conexión Ethernet, microHDMI, puerto USB, USB OTG (*On The Go*), ranura microSD, estéreo Jack (input y output), JITAG, Socket de alimentación (5 V), puerto para cámara, 2 buses CAN (*Controller Area Network*) y puertos de expansión.

BeagleBone tiene a su disposición diferentes sistemas operativos para trabajar. Los creadores de este sistema, Texas Instruments, han creado distribuciones de *Android* y *Linux*. Por otro lado, basados en Linux, están dispnibles *Angstrom Distribution, Ubuntu, Debian, ArchLinux, Gentoo, Sabayon, Buildroot, Erlang y Fedora*. Otros disponibles son *QNX y FreeBSD*.

Al igual que otros sistemas como Arduino o Raspberry Pi, tiene dos posibilidades de conexión a Internet; mediante cable Ethernet conectado en el puerto disponible en la placa y mediante WiFi, a partir de la conexión por USB de un *dongle* WiFi.

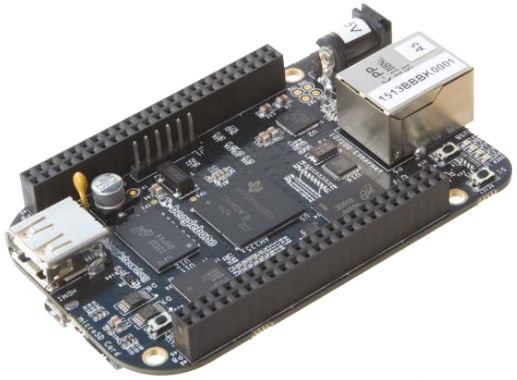

Ilustración 2.5. Vista BeagleBone Black.

2.2.5 Waspmote

Waspmote (ver Ilustración 2.6) es una plataforma Open Source, desarrollada por *Libelium*, optimizada en cuanto a consumo y aplicaciones. Pensada para la creación de grandes redes de sensores y para la comunicación en grandes distancias, buscando además gran autonomía del dispositivo. Es ideal para la creación de nodos en *Smart Cities*, ya que cuenta además con acelerómetro en 3 ejes para la detección de caídas. Además, soporta condiciones ambientales adversas (temperaturas entre -10 ºC y 65 ºC). [5]

Cuenta con un microcontrolador ATmega1281 de 14 MHz, memoria flash de 128 KB y posibilidad de inserción de tarjeta SD de hasta 2 GB. Dispone de reloj interno de tiempo real de 32 kHz. Puede ser alimentado con voltajes de entre 6 y 12 V, lo que es realmente útil cuando se utilizan paneles solares como fuente eléctrica.

Dispone de 7 puertos analógicos, 8 digitales I/O, 2 UART, una interfaz I^2C, SPI, USB, y cuenta además con sensores básicos de temperatura, humedad relativa y luminosidad (LDR).

El IDE de programación de Waspmote es muy parecido al de Arduino en cuanto a interfaz y utiliza exactamente la misma sintaxis, de forma que los sketches creados son compatibles entre los dos sistemas.

Siguiendo la filosofía de Arduino, cuenta con múltiples bloques de expansión añadiendo características muy interesantes, aunque principalmente se encuentran placas para implementación de sensores.

Una de las grandes características de Waspmote es su gran alcance en la comunicación. Cuenta con siete tipos de frecuencias para diferentes alcances que van desde los 2,4 GHz para 7 km hasta los 868 MHz para 40 km.

Ilustración 2.6. Vista de la placa Waspmote.

2.2.6 Nanode

Nanode (ver Ilustración 2.7) es una pequeña placa con conectividad únicamente Ethernet, que se está haciendo un hueco en el IoT, permitiendo conectar dispositivos a la red. Aunque con un diseño ligeramente distinto, sigue la misma filosofía que Arduino e incluso es compatible con algunas *shields*. Una de las cosas más interesantes es que puede enviar datos directamente a Pachube (actualmente Xively), por lo que es posible guardar gráficas de los valores obtenidos. Emplea un microcontrolador ATmega 328.

Usa exactamente el mismo IDE que Arduino utilizando también las mismas librerías. A diferencia de Arduino, no se puede conectar directamente por USB al PC para la compilación de los sketches, sino que se deben introducir desde el propio navegador en la web de Pachube.

Ilustración 2.7. Vista Nanode.

2.3 Plataformas del Internet de las Cosas.

Existen diversas plataformas o ecosistemas en la nube que permiten el almacenamiento, visualización, gestión y tratamiento, entre otras funcionalidades, de la información enviada. Estos ecosistemas van a permitir crear aplicaciones, tomando como fuente los datos enviados por los dispositivos, sensores y actuadores conectados a la red. Son una parte clave del *cloud computing*, en concreto en el ámbito de las *Smart Cities* e *Internet of Things*, dado que ayudarán a procesar la ingente cantidad de datos disponibles para obtener resultados de gran utilidad a la hora de crear proyectos.

Como se comentó en la introducción de este capítulo, cada vez más dispositivos se conectan a la red y en un futuro próximo serán miles de millones. Para los ecosistemas de desarrollo de aplicaciones web, este hecho va a suponer más información disponible y mayor cantidad de aplicaciones y servicios desarrollados.

Actualmente los servicios que se suelen prestar se enumeran a continuación:

- Almacenamiento. Se ofrece almacenamiento, limitado por número de Gigabytes, de archivos en la nube para tener acceso a ellos desde cualquier sitio.

- Informática. Existe la posibilidad de tener alojada en la nube una computadora, generalmente representada como máquina virtual (VM) o *instancia*. Es posible escalarla para tener mayor capacidad o menos.

- Bases de datos. Almacenar información en bases de datos ubicadas en un servidor remoto.

- Aplicaciones Web. Se trata del uso de software sin la necesidad de tener instalado el software en nuestro equipo.

- Análisis de datos. Cualesquiera que sean los datos enviados a la plataforma, pueden ser analizados y procesados para obtener informes, gráficas y resultados de utilidad.

- Servicios de Red. Es posible crear una red local virtual entre equipos que realmente están en lugares remotos

Al igual que ocurre con los sistemas empotrados hay disponibles diversas plataformas de desarrollo de aplicaciones en la nube. A continuación se van a describir las más importantes y relevantes en la actualidad.

2.3.1 Amazon Web Services (AWS)

Es una colección de servicios de computación en la nube lanzada en 2006, que en conjunto forman una plataforma de computación en la nube, ofrecidas a través de Internet por *Amazon.com*. Es usado en aplicaciones populares como *Dropbox, Foursquare, HootSuite*. Es una de las ofertas internacionales más importantes de la computación en la nube. [6]

AWS está situado en 9 Regiones geográficas: EE.UU. Este (Norte de Virginia), EE.UU. Oeste (Norte de California), EE.UU. Oeste (Oregón), AWS GovCloud (EE.UU.), São Paulo (Brasil), Irlanda, Singapur, Tokio y Sydney. Cada región está totalmente contenida dentro de un solo país y todos sus datos y servicios permanecen dentro de la región designada.

Presta servicios de informática con instancias de Linux, cuyo coste se evalúa por horas de uso, almacenamiento casi ilimitado con una tarifa por GB (*Amazon Cloud Drive*), bases de datos SQL , Redes y CDN, análisis de datos almacenados y servicios de aplicaciones para software ejecutándolo en *streaming* desde la nube.

Se denomina *Amazon EC2* (Amazon *elastic compute cloud*) a la parte central de la plataforma de cómputo en la nube de la empresa Amazon.com. EC2 permite a los usuarios alquilar PCs virtuales, en los cuales poder correr sus propias aplicaciones

Se trata de una plataforma muy desarrollada, ofrece gran cantidad de servicios adaptadas a diferentes exigencias y es fácil de implementar. En contra, la mayoría de servicios son de pago, aunque algunos sean adquiribles de forma gratuita.

En la Ilustración 2.8 se muestra la interfaz del portal de AWS.

Ilustración 2.8. Vista del portal de Amazon Web Services.

2.3.2 Google Cloud Platform

Google Cloud Platform (ver Ilustración 2.9) es la plataforma de Google de *Cloud computing*. Sus inicios se remontan a abril de 2008, cuando por primera vez apareció *Google App Engine*. Finalmente, ha sido lanzada en marzo de 2014 con un gran elenco de servicios a disposición del usuario. [6]

Los servicios que presta esta plataforma, entre otros de menor importancia para este proyecto, son:

- *Google Compute Engine,* aplicaciones Web de recursos escalables.
- *Google App Engine*, informática en la nube para creación de VM.
- *Google Cloud Storage* para almacenamiento en la nube.
- *Google Cloud SQL* gestor de bases de datos.
- *Google BigQuery* análisis de datos en Big Data.

Dada la creciente aparición de este tipo de plataformas, las empresas deben ofrecer características distintas a las de las demás. En el caso de *Google Cloud Platform*, cuenta con una gran red propia de fibra óptica que se extiende por medio mundo, y que va a permitir a los clientes obtener una gran velocidad de transferencia de datos al usar sus servicios. Además, se destaca el hecho de que todo está montado sobre los servidores de Google, que a su vez están conectados a dicha red y que va a hacer el sistema mucho más rápido y fiable. Google Cloud Platform pone además varias API's a disposición de los usuarios para la creación de las aplicaciones web, de forma que cada cual puede crear sus propios proyectos.

Al igual que ocurre con Amazon Web Services, *Google Cloud Platform* cuenta con diferentes posibilidades en sus servicios, que en la mayoría de los casos serán de pago.

Ilustración 2.9. Vista del portal de Google Cloud Platform.

2.3.3 Fi-Ware

La Comisión Europea (CE) y las principales empresas TIC europeas emprendieron en 2011 un programa de Colaboración Público-Privada (*Public Private Partnership* – PPP), con el objetivo de definir una plataforma que represente una opción abierta para el desarrollo y despliegue global de aplicaciones en la Internet del Futuro. La tarea fue encomendada a Telefónica, la cual creó Fi-Ware. Las especificaciones de las APIs (*Application Programming Interfaces*) ofrecidas por los componentes de esta plataforma son abiertas y completamente libres de royalties. En la actualidad se está implementando en grandes ciudades de Europa para la creación de *Smart Cities*. [7]

Los principales servicios prestados por esta plataforma son: elementos de Cloud Hosting, que proporcionan los recursos básicos de computación, de red y de almacenamiento; elementos de manejo de datos, que proporcionan herramientas para análisis tipo "Big Data"; módulos para integración de aplicaciones, que proporciona elementos para integrar aplicaciones, permitir su publicación, etc., así como aspectos de negocio, elementos para el manejo de la sensores e Internet de las cosas, que permiten utilizar de forma fácil y estandarizada los recursos de los sensores y actuadores; módulos para el acceso a la red de comunicaciones y control de terminales, y elementos para dotar de seguridad y privacidad a las aplicaciones.

En general, se denominan *General Enablers* a los diferentes servicios disponibles en su catálogo. Posee un entorno web de *Cloud* donde es posible enlazar unas aplicaciones o widget con otras, hasta conseguir otras nuevas de mayor complejidad sin tener que crearlas desde cero.

Fi-Ware está en continuo desarrollo, promueve la creación de nuevos proyectos e ideas por parte de desarrolladores y usuarios, y a diferencia de la mayoría de ecosistemas en la nube, es completamente abierto y gratuito.

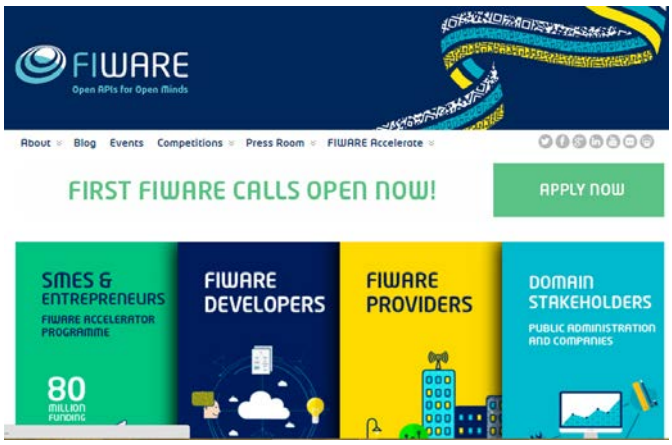

Ilustración 2.10. Vista de la web principal de FI-WARE.

2.3.4 Xively

Xively (ver Ilustración 2.11) es una división de *LogMeIn Inc,* que ofrece una plataforma de Internet de las Cosas como servicio para empresas y usuarios. [6]

En 2007, fue fundada *Pachube,* una infraestructura de Internet de las Cosas, en Londres. En 2011 se anunció que habían sido comprados por LogMeIn y se pasó a llamar *Cosm*. Finalmente, en Mayo de 2013, dado el poco éxito de Cosm se volvió a renombrar y pasó a llamarse *Xively*, que además pasó a ser pública.

Principalmente cuenta con los siguientes servicios:

- *Xively Cloud Services*, construido para Internet de las Cosas. Incluye servicios de datos, un servicio de seguridad Trust Engine y aplicaciones gestionadas desde la web. La comunicación con Xively está construida mediante protocolos publish-subscribe llamados MQTT. El API soporta REST, WebSockets y MQTT.

- *Xively Business Services*, el cual va orientado a los servicios y aplicaciones para empresas.

- *Xively Partner Network:* es una asociación con compañías de chipsets como ARM, Atmel y TI como proveedores, así como alianzas con compañías de Internet de las Cosas como OASIS.

Al igual que otras plataformas, cuenta con API's que permiten al usuario crear aplicaciones Web, conectar dispositivos y gestionar los datos enviados y recibidos desde ella. Xively está muy desarrollada y permite realizar todas estas tareas de forma rápida y sencilla. Muestra de ello es la consola Web para la gestión de todos los dispositivos con la que cuenta, donde se muestra información detallada de todos ellos.

Ilustración 2.11. Vista del portal web de Xively.

2.3.5 Microsoft Azure

Microsoft Azure (denominada anteriormente *Windows Azure y Azure Services Platform*) es una plataforma alojada en los Data Centers de Microsoft. Anunciada en 2008 en su versión beta, pasó a ser un producto comercial el 1 de enero del 2010. *Windows Azure* es una plataforma general que proporciona diferentes servicios para aplicaciones, desde servicios que alojan aplicaciones en alguno de los centros de procesamiento de datos de Microsoft, para que se ejecute sobre su infraestructura (Cloud Computing), hasta servicios de comunicación segura y federación entre aplicaciones. [6]

El servicio de proceso de Windows Azure ejecuta aplicaciones basadas en Windows Server. Estas aplicaciones se pueden crear mediante .NET Framework en lenguajes como C# y Visual Basic, o implementar sin .NET en C++, Java y otros lenguajes.

Windows Azure utiliza un sistema operativo especializado, llamado de la misma forma, para correr sus "capas" (en inglés "*fabric layer*"). Un clúster localizado en los servidores de datos de Microsoft, que se encarga de manejar los recursos almacenados y el procesamiento para proveer los recursos (o una parte de ellos) para las aplicaciones que se ejecutan sobre Windows Azure.

Los servicios prestados por Windows Azure son:

- *Windows Azure Compute:* es una plataforma para hospedar y administrar aplicaciones en los centros de datos de Microsoft.

- *Windows Azure Storage:* tiene servicios de básicos como parte de la cuenta de almacenamiento.

- *Microsoft SQL Azure:* es un servicio de base de datos en la nube basado en las tecnologías de SQL Server.

- *Content Delivery Network* (CDN): coloca copias de los datos cerca de donde estos se encuentran.

- *Azure AppFabric:* ofrece diferentes servicios para aplicaciones. Los servicios de autenticación, autorización y mensajería permiten la comunicación segura entre aplicaciones y servicios desplegados tanto en la nube y en local.

- *Azure Market Place* es un mercado en línea global compartir, comprar y vender aplicaciones SaaS completas y conjuntos de datos.

- *Azure Virtual Network* es una serie de funciones de red.

Ilustración 2.12. Vista del portal de Microsoft Azure.

2.3.6 SensorCloud

SensorCloud es una plataforma de *MicroStrain* que se centra en el almacenamiento, visualización y gestión remota de datos. Es muy potente y permite el procesamiento en la nube de la información proporcionando gran escalabilidad de los datos, rápida visualización y análisis programados por el usuario. [6]

Originariamente se creó para dar soporte a despliegues de sensores inalámbricos de MicroStrain sin embargo, ahora soporta cualquier dispositivo físico, sensor o red de sensores a través de una simple API OpenData.

Como ya se ha mencionado esta plataforma está optimizada para el manejo de datos, y es por ello, que suministra gran cantidad de servicios para el manejo, análisis y control de sensores de todo tipo, pudiendo recibir notificaciones de múltiples formas: SMS, correo electrónico, notificaciones en Smartphone, etc.

Mathengine es un potente motor de cálculo ofrecido por esta plataforma para operar sobre los datos representados gráficamente de forma instantánea, de forma que es posible filtrar, suavizar, aplicar transformaciones (FFT's), todo ello en tiempo real.

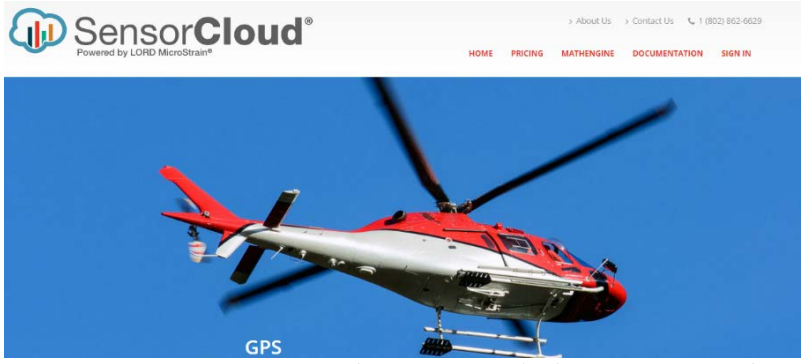

Ilustración 2.13. Vista del portal web de SensorCloud.

Capítulo 3

Descripción del sistema

3.1 Introducción

En el capítulo anterior se han estudiado algunos los dispositivos empotrados y plataformas de Internet de las Cosas actualmente disponibles.

Dentro de las alternativas propuestas, se ha seleccionado FI-WARE como plataforma y como dispositivos empotrados para integrar en la plataforma, Arduino y Raspberry Pi.

Se ha decidido utilizar FI-WARE por varias razones. En primer lugar, ha sido seleccionada como la plataforma europea para las Ciudades Inteligentes e Internet de las Cosas. En segundo lugar, es una plataforma totalmente gratuita. Se trata de una plataforma innovadora que tiene gran potencial, promueve el emprendimiento y fomenta la creación de nuevos proyectos en varios continentes.

En cuanto a los dispositivos empotrados, se han seleccionado Arduino UNO y Raspberry Pi modelo B. Arduino es un sistema muy económico cuyo uso está muy extendido en el mundo académico, es fácil de programar, con grandes especificaciones y de código abierto. Existen multitud de librerías que añaden funciones pre-programadas, por ejemplo, para el uso de HTTP, Wifi o Tarjeta SD, que hacen más sencilla la interacción con FI-WARE. Además, los Shields disponibles para aportar nuevo hardware al sistema son de gran utilidad al permitir, junto con FI-WARE, la creación de diferentes y novedosas aplicaciones y servicios.

Raspberry Pi, por otro lado, se trata de otra opción de gran interés y es por ello que también ha sido elegida para formar parte de este proyecto. Al igual que Arduino, se trata de un dispositivo muy económico, de uso muy extendido y open source. Esto hace que haya mucha información, proyectos y librerías por la web. Está además basada en Linux, por lo que es posible compilar programas de dicho sistema operativo. Todo esto sumado a que se pueden encontrar elementos que extienden su funcionalidad (cámaras, baterías, alimentación solar, entre otros) hacen de Raspberry Pi un sistema ideal para incorporar en Ciudades Inteligentes.

3.2 FI-WARE

En el capítulo anterior se han resumido las características de esta plataforma brevemente. A continuación se va a describir el sistema con más detalle, profundizando en los elementos y características que serán necesarios en la realización de este proyecto.

Los puntos que se van a ir desarrollando sobre diferentes aspectos de la plataforma se resumen a continuación:

- Arquitectura de FI-WARE, cómo está constituido el sistema.
- FI-WARE está construida con elementos conocidos como Generic Enablers (GE's).
- FI-LAB es la instancia de FI-WARE que materializa el ecosistema.
- Herramientas que FI-WARE pone a disposición de los desarrolladores para crear aplicaciones.

3.2.1 Arquitectura

La arquitectura de FI-WARE se basa de forma general en un conjunto de bloques interconectados y que operan de forma conjunta. En la Ilustración 3.1 se muestran dichos bloques y sus relaciones [8].

Herramientas de la arquitectura y comunidad de desarrolladores (Developer Community and Tools Architecture).

El principal objetivo de la Comunidad de desarrolladores (DCT) es ofrecer un ecosistema de desarrollo multi-funcional (FI-CoDE), permitiendo la creación y gestión de las futuras aplicaciones de Internet de las Cosas (FIApp). La comunidad ha sido creada para guiar las necesidades del Internet del futuro basándose en la integración de nuevos GEs.

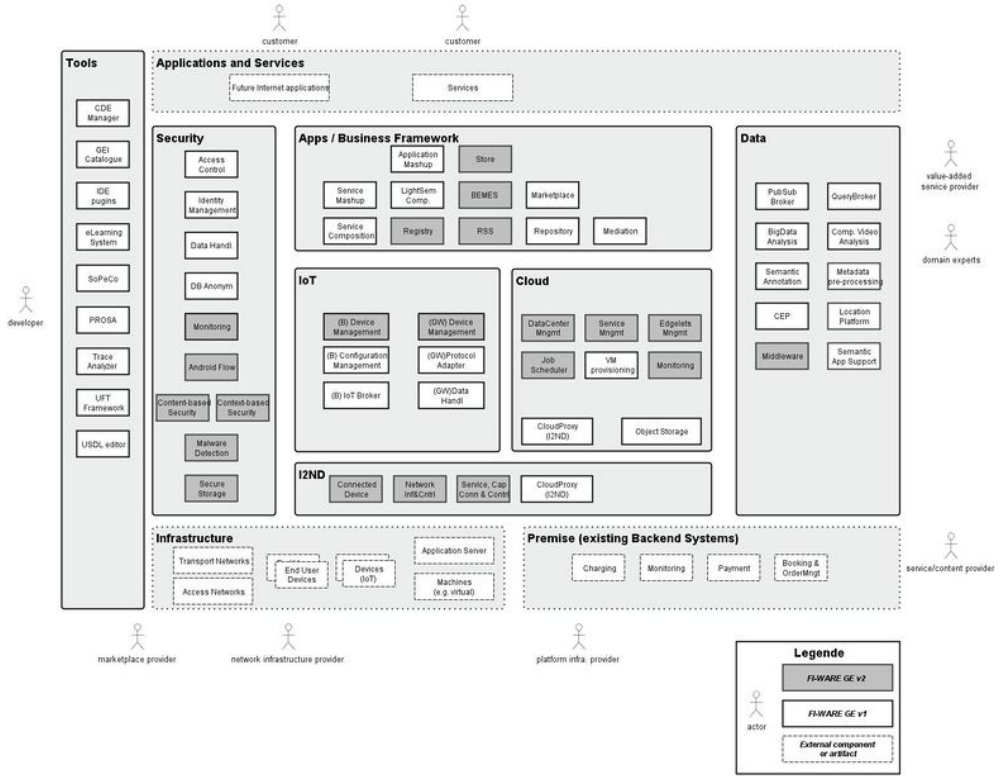

Ilustración 3.1. Esquema general de la arquitectura de FI-WARE.

Ecosistema de Aplicaciones/Servicios y marco de entrega (Applications/Services Ecosystem and Delivery Framework)

Los GEs para la Apps Chapter forman juntos un ecosistema de aplicaciones y servicios sostenible e innovador. En particular, el App Generic Enabler soporta la gestión de servicios en el marco de las empresas a través de todo el ciclo de vida del servicio, desde la creación/composición del servicio hasta la monetización y recuperación de la inversión.

Gestión de datos de contexto (Data/Context Management)

Esta sección tiene como objetivo proporcionar plataformas como GEs que faciliten el desarrollo y la creación de aplicaciones innovadores que requieren la recopilación, publicación, procesamiento y explotación de información y datos obtenidos en tiempo real y a escala masiva. Combinado con GEs de la sección de App, los proveedores de aplicaciones serán capaces de construir modelos de negocio innovadores basados en la explotación de los datos masivos facilitados por los usuarios finales.

FI-WARE Data/Context Management GEs permite recibir información del context y otras fuentes (Context Broker), mediar con GEs y usarla en aplicaciones a través de la capa homogénea Query Broker, donde se puede trabajar con esa información, añadirla a la ya existente, almacenarla, etc. Se permite incluso la sencilla creación de aplicaciones de vídeo en *streaming* mediante el *Stream Oriented* GE y *Video Analysis in the Compressed Domain* (CDVA).

Interfaces para redes y dispositivos (Interface to Networks and Devices, I2ND)

I2ND define un espacio para proveer de GEs para implementar y ejecutar una infraestructura de red estandarizada. La infraestructura tendrá que competir con terminales muy sofisticados, además de sofisticados proxis por un lado y con operadores de la red por el otro. Más tarde esto será implementado físicamente mediante nodos, bajo el control de un operador.

La arquitectura I2ND cubre los siguientes Generic Enablers (GEs): CDI (Connected Device Interface) alrededor de los dispositivos conectados, CE (Cloud Edge) a través de los proxis en la nube, NETIC (NETwork Information and Control) a través de redes abiertas and S3C (Service Capability, Connectivity and Control) a través de redes subyacentes.

Habilitador de servicios de Internet de las Cosas (Internet of Things (IoT) Services Enablement)

El despliegue de la arquitectura del Internet de las Cosas está distribuido a través de un gran número de dispositivos, Gateways y Backends.

Un dispositivo (device) es una entidad de hardware, componente o sistema que mide propiedades de una cosa o grupo de cosas o influencias de propiedades de una cosa o grupo de cosas. A modo de ejemplo, sensores y actuadores son dispositivos.

Los recursos de IoT son elementos computacionales (software) que aportan significado técnico a la sensorización y actuación de un dispositivo. El recurso está habitualmente alojado en el dispositivo.

Los GEs de IoT han sido instanciados en dos dominios diferentes: Gateway y Backend.

Almacenaniemto en la nube (Cloud Hosting)

Se ofrecen GEs que permiten diseñar un moderno servicio de hosting que puede ser usado para desarrollar, lanzar y gestionar aplicaciones y servicios del Internet del Futuro (FI).

El GE IaaS Data Center Resource Management (DCRM) ofrece aprovisionamiento y gestión del ciclo de vida de los recursos virtualizados (computación en la nube, almacenamiento, redes) asociados a las máquinas virtuales. El GE Object Storage ofrece almacenamiento basado en objetos. El GE Job Scheduler ofrece aplicaciones para añadir y gestionar trabajos virtuales computacionales de forma unificada y escalable. El GE Edgelet Management ofrece la capacidad de almacenar componentes de aplicaciones de poco peso o tamaño. Finalmente, el GE PaaS Managenment usa las características anteriores para aportar

gestión completa de los ecosistemas de PaaS, el desarrollo y configuración de software (SDC), que ofrece instalaciones y personalizaciones de software de forma flexible en Chef récipes dentro de las máquinas virtuales.

Seguridad (Security)

En resumen la arquitectura de seguridad de FI-WARE busca demostrar que la visión de un Internet que es seguro por diseño se está convirtiendo en una realidad. La privacidad y la confianza en FI-WARE está principalmente concentrado en el desarrollo de herramientas y técnicas para obtener la seguridad mencionada.

La arquitectura de alto nivel referenciada está formada por cuatro bloques principales:

- Generic Security Services
- Acces Control
- DB anonymizer
- Malware Detection Service

3.2.2 Generic Enablers

Los GEs son la base fundamental de FI-WARE, tal y como se ha mencionado anteriormente. En el catálogo hay disponible una extensa lista de GEs (ver Tabla 3.1), sin embargo se detallarán los de mayor importancia para el objetivo final de este proyecto. [9]

Generic Enablers	
Access Control - THA Implementation	Object Storage GE
Application Mashup - Wirecloud	PaaS Manager - Pegasus
Backend Device Management - IDAS	Protocol Adapter - MR CoAP
BigData Analysis - Cosmos	Publish/Subcribe Context Broker
Complex Event Processing (CEP)	Repository - SAP RI
Configuration Manager - Orion CB	Revenue Settlement and Sharing System
Gateway Data Handling GE	Self-Service Interfaces - Cloud Portal
IaaS Data Center Resource Manage	Software Deployment & Config
Identity Management - KeyRock	Store - Wstore
NEC IoT Broker	Stream-oriented - Kurento

Tabla 3.1. Listado de los GEs de FI-WARE.

3.2.2.1 Application Mashup – Wirecloud

Wirecloud permite el desarrollo de aplicaciones web destinadas al usuario final mediante una interfaz intuitiva y vanguardista. Mediante la utilización de la información disponible permite la creación final de los servicios del Internet del Futuro.

Los *Mashups* o aplicaciones web integran masas heterogéneas de datos, lógica mediante aplicaciones y componentes de interfaz de usuario (UI), como son los *Widgets/Gadgets* y los *Operators* (widgets sin interfaz, sólo lógica). Permite que los widgets y operadores puedan ser enlazados para crear composiciones más complejas. Se promueve que todas las creaciones sean compartidas entre usuarios y estén disponibles para todos. Para ello, se dispone de mercado de aplicaciones llamado *Marketplace,* además de un *Store* para la publicación de servicios Web. En la Ilustración 3.2 se puede ver un ejemplo la UI y el *wiring* entre widgets. En dicho ejemplo se enlazan un widget de entrada de texto con un reproductor de Youtube, de forma que se puedan realizar búsqueda de vídeos.

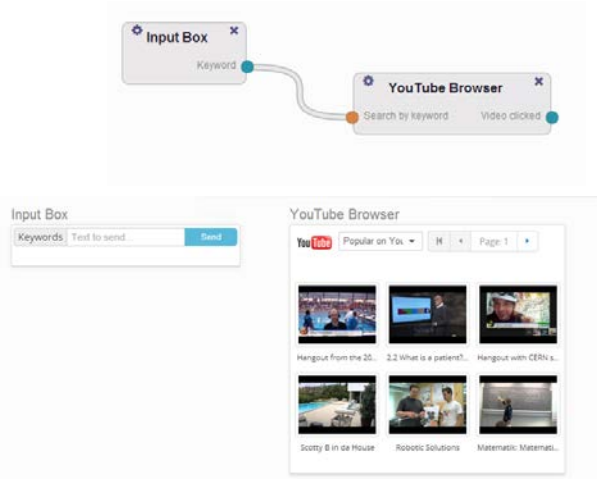

Ilustración 3.2. Ejemplo Wiring y UI

La creación de widgets se realiza utilizando maquetación web con lenguajes como HTML, CSS, Javascript y XML, y empleando la API NGSI que FI-WARE pone a disposición del desarrollador para acceder a la información de contexto.

La estructura principal de un widget debe contener los siguientes archivos:

- Index.html. Donde se crea el diseño del widget y los elementos que lo conformarán.

- Config.xml. Es un archivo XML donde se detallan los siguientes aspectos:

    - Datos del creador tales como nombre, dirección, e-mail, etc.

- Preferencias de la plataforma.
- Imagen del widget en las distintas plataformas.
- Entradas y salidas (I/O) para poder crear mashups mediante wiring.
- Tamaño del widget.

- Javascript. Es un directorio donde se guardarán todos los documentos .js que sean necesarios. En ellos es donde se configura la conexión con FI-WARE.
- CSS. Directorio donde se guardan las hojas de estilos del aspecto del widget.
- Images. Directorio donde se pueden guardar las imágenes que son usadas como recurso en el diseño.

Finalmente, para poder utilizar el widget en la sección Mashup de FI-WARE es necesario comprimir los archivos en formato *.zip* cambiando la extensión de éste a *.wgt,* posteriormente.

3.2.2.2 Publish/Subscribe Context Broker – Orion Context Broker

Orion Context Broker es una implementación del GE *Public/Subscribe Context Broker*, mediante las interfaces *NGSI-9* y *NGSI-10*. Gracias a ellas los usuarios pueden realizar diferentes operaciones:

- Registrar aplicaciones productoras de información de contexto, p. ej. un sensor de temperatura en una habitación.
- Actualizar el valor de contexto, p. ej. actualizar el valor del sensor de temperatura.
- Recibir notificaciones cuando se produce un cambio en la información de contexto o cada cierto tiempo, p. ej. recibir el valor de temperatura cada minuto.
- Consultar la información de contexto mediante *querys*.

NGSI-9 y NGSI-10 son Interfaces de programación de Aplicaciones (APIs) de código abierto que pueden ser descargadas desde la web de FI-WARE. Dichas APIs permiten la Transferencia de Estado Representacional (Representational State Transfer, REST) y sus operaciones *GET y POST*. Por tanto, la comunicación se realiza por HTTP, para lo cual se pueden emplear dos posibles formatos de datos: *JSON* y *Formularios XML* (solamente XML si se utiliza NGSI9). En la Ilustración 3.3 se muestra cómo se estructura la

información con cada API dentro del servidor donde se alojen las instancias y las principales operaciones.

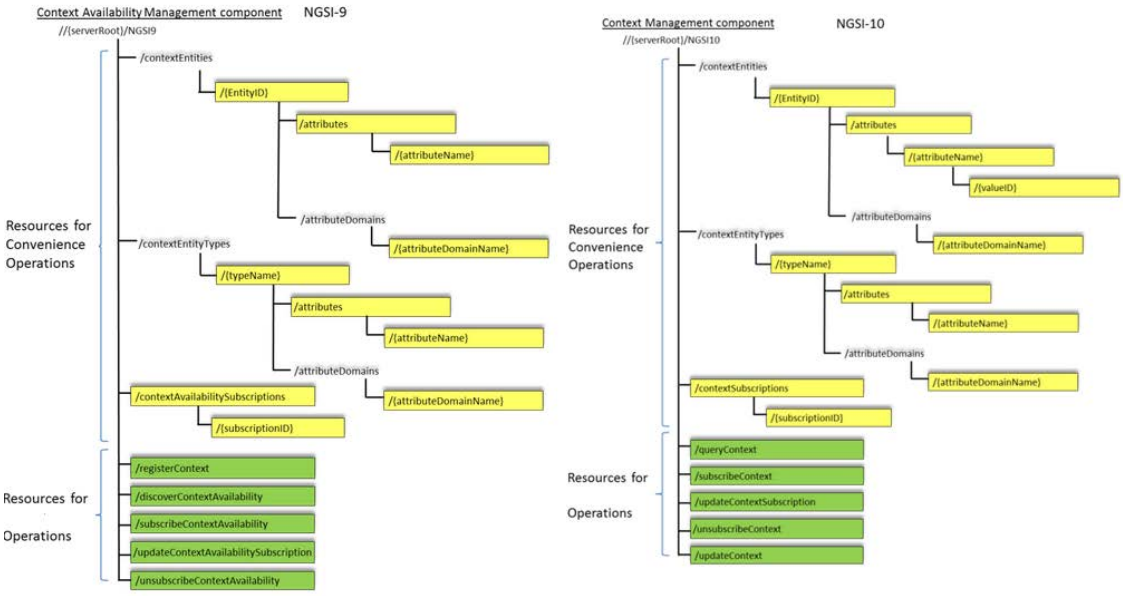

Ilustración 3.3. Estructura y operaciones NGSI9 izquierda y NGSI10 derecha.

La diferencia principal entre las APIs es que NGSI9 trabaja con la disponibilidad de la información de contexto mientras que NGSI10 trabaja con la propia información, permitiendo consultar valores. De esta forma, NGSI9 permite conocer datos de identificadores y atributos pero no sus valores de contexto. Se puede apreciar también una diferencia en la sintaxis de ambas a la hora de llevar a cabo las diferentes operaciones.

Será dentro de las instancias donde se ubicará la información de contexto. Ésta información a su vez se organiza en forma de *entidades o entities*. Las entidades tienen una serie de características propias que son cruciales a la hora de subir, actualizar o consultar la información:

- Tipo de entidad o *entity type*. Define el tipo de entidad, generalmente sensor, actuador, etc.

- Nombre o *Id*. Es el identificador de dicha entidad, es único para cada entidad de una misma instancia.

- Atributos o *Attributes*. Son los diferentes atributos que se guardarán dentro de la entidad. Pueden ser uno o varios y cada uno tiene a su vez otras dos etiquetas: tipo y valor de contexto (*type y context value*). Ejemplo de atributo puede ser: atributo temperatura, tipo centígrado y valor de contexto 23 ºC.

Existen dos tipos generales de Context Broker. Uno de ellos, es el que se denomina *propio* es aquel al que sólo tiene acceso el desarrollador que lo ha instanciado. El segundo tipo, es el que se denomina *global*. Este tiene la característica de ser accesible por todos los usuarios, tanto para añadir información, como para consultarla de forma que se puede hablar de *Open Data* cuando se trabaja en ella. La instancia de este Context Broker es http://orion.lab.fi-ware.org

En el Anexo I se desarrolla de forma extendida cómo trabajar con este Generic Enabler, así como las herramientas de trabajo que permitirán posteriormente crear un caso de estudio. La documentación ahí disponible es válida para la configuración de *un Context Broker* propio. En el caso de utilizar la instancia global debería utilizarse la documentación disponible en https://forge.fi-ware.org/plugins/mediawiki/wiki/fiware/index.php/Publish/Subscribe_Broker_-_Orion_Context_Broker__Quick_Start_for_Programmers

3.2.2.3 BigData Analysis – Cosmos

Cosmos es una implementación del GE Big Data que permite el despliegue de clusters de computación privada basados en el ecosistema *Hadoop*. La versión actual de Cosmos permite los usuarios:

- Operaciones de entrada y salida (I/O) mediante *Infinity,* un cluster de almacenamiento persistente basado en HDFS.

- Creación, uso y eliminación de clústeres basados en *MapReduce* y bases de datos del tipo SQL como Hive o Pig.

- Gestionar la plataforma, en muchos aspectos como servicios, usuarios, clústeres, etc., desde la API de Cosmos o desde CLI.

- También existe un componente llamado *Cygnus* encargado de recibir información de Contexto desde Orion y almacenarlo en HDFS.

En definitiva, este Generic Enabler permite al usuario crear bases de datos con la información recibida, de forma que no sólo esté disponible el último valor enviado por un dispositivo, sino una completo listado temporal con el que poder hacer análisis de mayor complejidad. Además, si se realiza el almacenamiento mediante bases de datos de tipo SQL, será posible hacer análisis de BigData mediante querys complejas que permitan filtrar los datos y conseguir resultados importantes de cara a la creación de servicios web y aplicaciones.

Cosmos dispone de un servidor, el cual es su instancia principal, en http://cosmos.lab.fi-ware.org.

En el Anexo I se muestra el proceso necesario para la configuración del sistema para hacer persistir datos en Cosmos.

3.2.3 FI-Lab

FI-Lab es el portal de FI-WARE desde donde los desarrolladores encuentran las principales herramientas necesarias, tanto para crear y gestionar la infraestructura de sus proyectos, como para la utilización y wiring de Widgets, además de la publicación de aplicaciones y servicios en el Market.

Este portal se divide en diferentes secciones que serán detalladas a continuación.

3.2.3.1 Cloud

La sección de *Cloud* (ver Ilustración 3.4) es aquella en la que se permite la creación, instalación de software y mantenimiento de las máquinas virtuales (VM), con las que los desarrolladores trabajan. Los *containers* que permiten el almacenamiento en la nube también se encuentran en este lugar. Además, se permite el despliegue de máquinas virtuales de diferentes capacidades e instalar sobre ellas software disponible en el catálogo de FI-WARE.

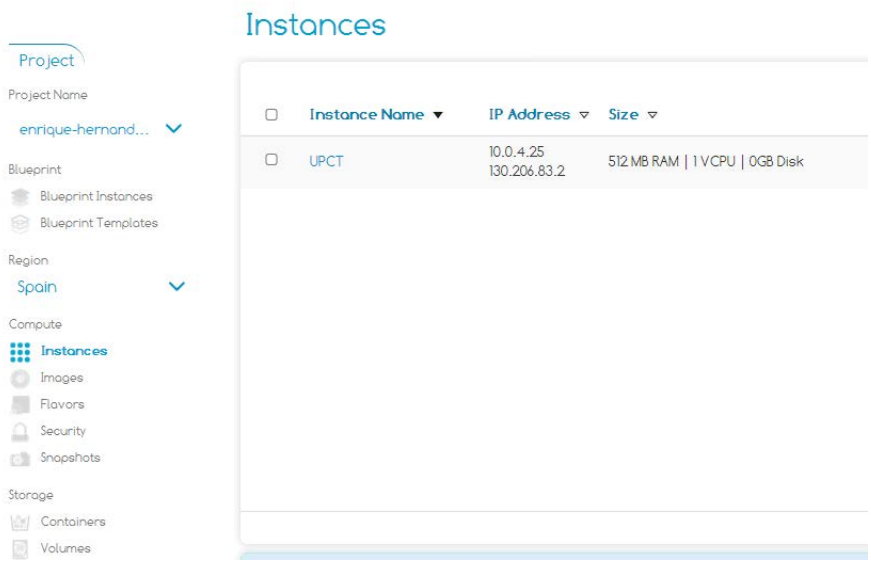

Ilustración 3.4. Vista de la sección de Cloud de FI-Lab

Una de las principales características de la creación de las VM es que sobre ellas se puede instalar software de dos formas distintas:

- Creando máquinas trabajando con software distintos en distintos niveles o capas (*Tiers)* y usando *Blueprint Templates* (ver Ilustración 3.5). Es decir, se utilizarán plantillas de software que mediante unos sencillos pasos instalarán y configurarán la VM de forma automática.

En la base debe encontrarse un software Linux y los que están disponibles en la web son CentOS y Ubuntu. Tras la instalación de esa base y selección del tamaño de VM se procede a la instalación del software restante. Es importante señalar que éste tipo de instalación hace a la VM auto escalable, de forma que en función de la carga del sistema empleará más o menos núcleos de procesamiento, pudiendo seleccionarse el número máximo y mínimo de éstos en la instalación.

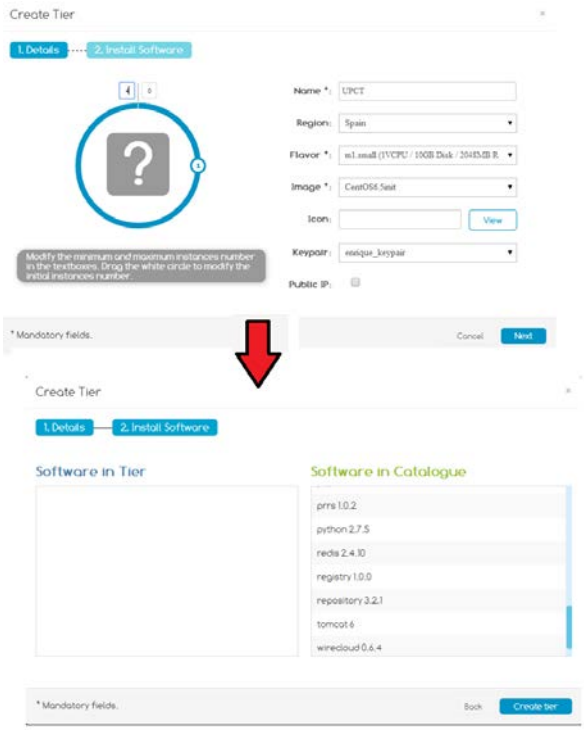

Ilustración 3.5. Proceso instalación mediante Tier y Bluetemplates.

- Existen imágenes pre-configuradas que permiten una instalación más rápida y sencilla (ver Ilustración 3.6). Existe, por ejemplo, una configuración rápida de un Context Broker mediante el lanzamiento de la imagen orion-psb-image.

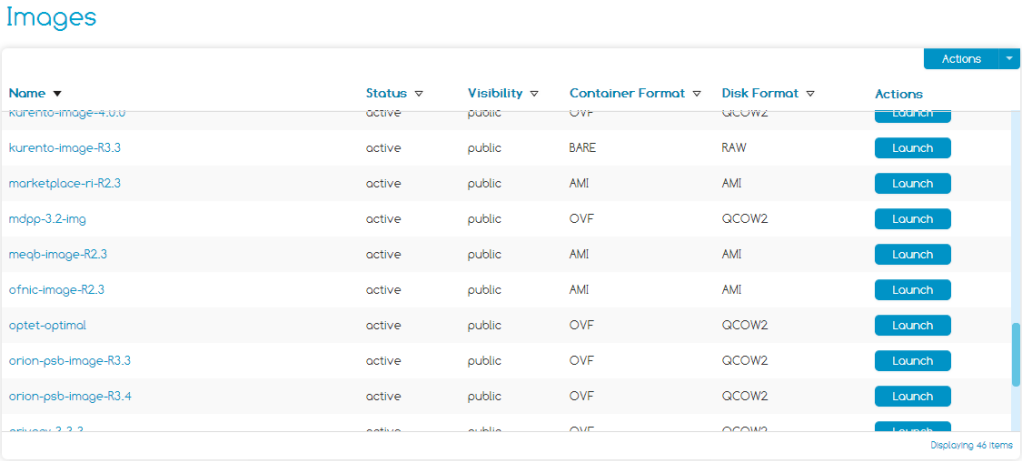

Ilustración 3.6. Vista instalación mediante imágenes

Las máquinas virtuales pueden ser de tres tamaños y potencias distintos, estos están reflejados en la Tabla 3.2.

Nombre	m1.tiny	m1.small	m2.largedisk
VCPU	1	1	2
RAM (MB)	512	2048	2048
Root Disk (GB)	0	10	20
Ephemeral Disk (GB)	0	20	40

Tabla 3.2: Capacidades de las VM disponibles para desplegar

El aspecto de la seguridad es algo que también se ha solucionado de forma sencilla. Existe un apartado donde se crean las IPs públicas, a través de las cuales se puede acceder a una máquina virtual y otro, donde se crean los llamados Grupos de Seguridad (*Security group),* en los cuales se permiten administrar los puertos abiertos a una máquina. Para el acceso a las máquinas a través de la IP pública se generan claves públicas llamadas *keypair*. Todo ello es asignable a una máquina virtual de forma intuitiva y en el caso de los keypair y security group, a varias a la vez.

Por último, en el entorno Cloud dispone en la parte inferior de la interfaz una consola de errores de utilidad a lo largo del proceso de desarrollo.

3.2.3.2 Store

En este apartado del portal FI-Lab se puede encontrar una tienda en la que se publican las aplicaciones que los desarrolladores han publicado. Éstos pueden ser gratuitos o de pago.

En la mayoría de los casos es posible descargar el código y observar su composición. Todo ello gracias a la filosofía de Código abierto de FI-WARE.

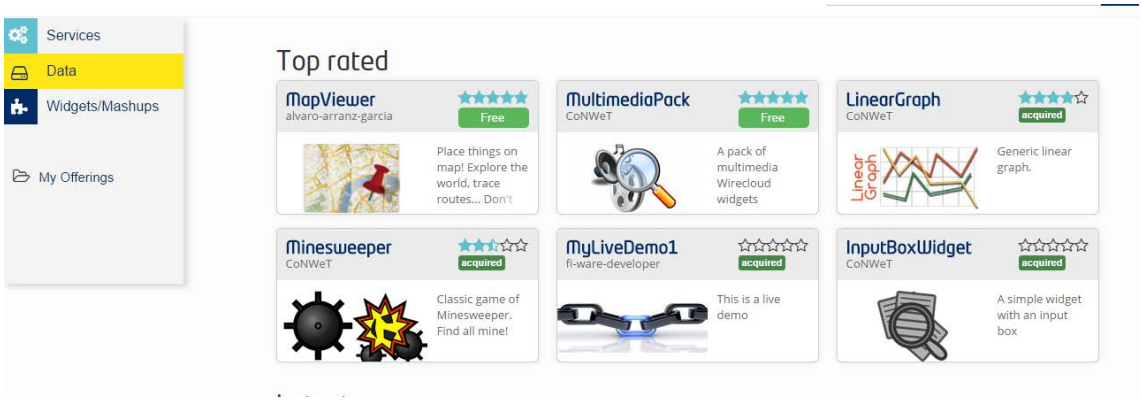

Ilustración 3.7. Vista apartado Store

3.2.3.3 Mashup

La sección *Mashup* es aquella donde se suben los widgets o se descargan otros existentes, enlazarlos con el ya conocido wiring y finalmente, hacerlos funcionar. Esta sección a su vez se divide en tres:

- *Marketplace*, aquí se pueden ver las aplicaciones disponibles, subir las propias e instalar y desinstalar del espacio de trabajo del usuario. Permite buscar aplicaciones por categorías y realizar filtros.

- *Wiring* (ver Ilustración 3.8), espacio para situar los widgets que se desean enlazar y crear las convenientes conexiones. Todo ello se realiza de forma intuitiva y gráfica. Además, se podrán configurar los widgets mediante el menú destinado a ello que cada uno posee.

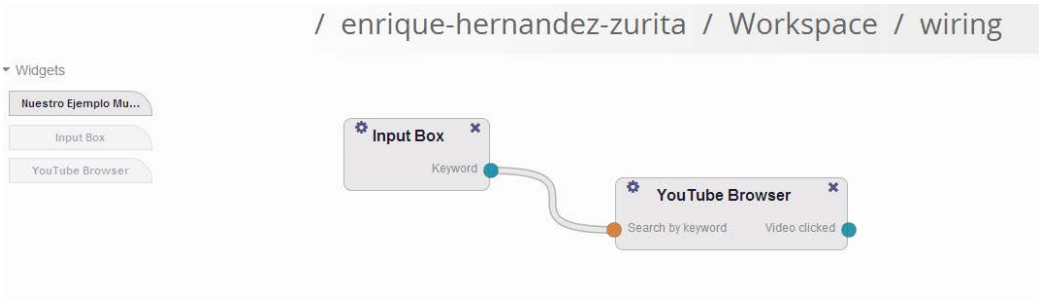

Ilustración 3.8. Vista sección Mashup/Wiring

- *Editor* (ver Ilustración 3.9), es el espacio donde ejecutar las aplicaciones y ver su funcionamiento final. Se subdivide en espacios de trabajo o *workspaces* con el fin de manejar diferentes widgets y aplicaciones en escenarios separados.

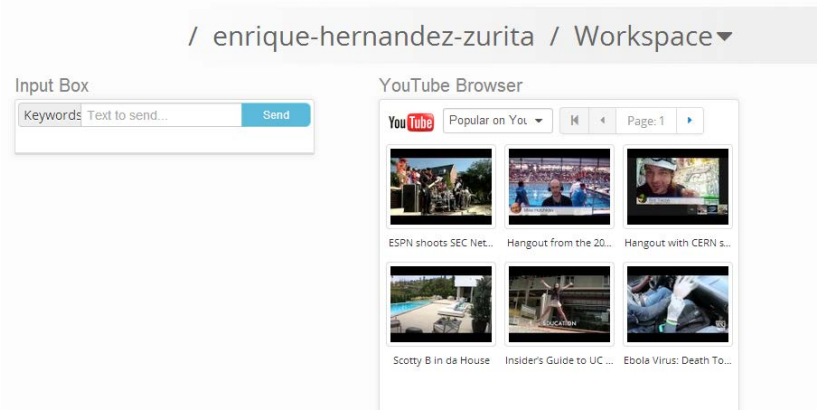

Ilustración 3.9. Vista sección Mashup/Wiring

3.2.3.4 Data

Este apartado contiene un portal de Open Data (ver Ilustración 3.10), que proporciona información real de ciudades inteligentes a los desarrolladores para que puedan experimentar con ellas. Actualmente, existen más de 950 hojas de datos de 8 organizaciones distintas.

Una de las grandes virtudes de este portal es la capacidad de buscar información mediante el uso de *tags* o temas más utilizados. Los datos pueden ser organizados por ciudades, organizaciones, etc. Además, pueden ser descargados en distintos formatos para facilitar su utilización.

Es posible publicar hojas propias, hacerlas privadas y cobrar por su utilización, ya sea pagando por descarga, suscripciones temporales o pago en función del uso.

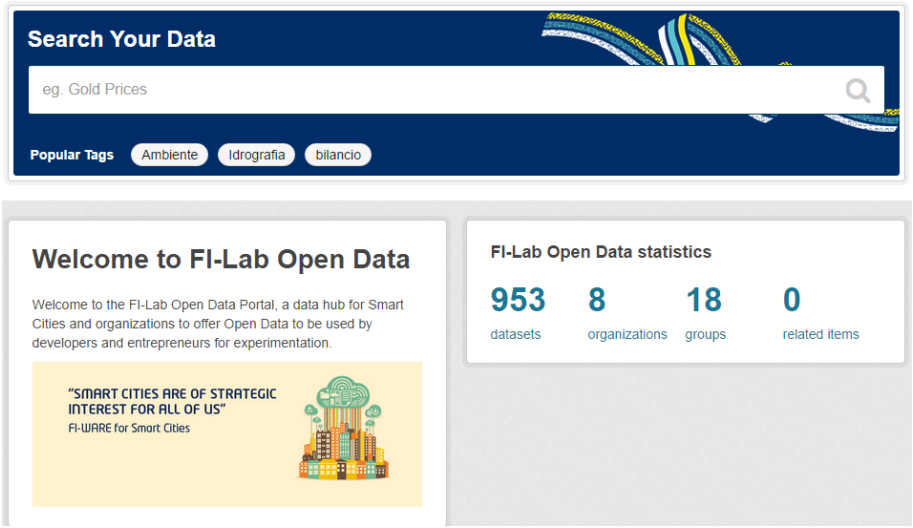

Ilustración 3.10. Vista del portal de Open Data de FI-WARE

3.2.4 Herramientas (Tools)

FI-WARE ofrece herramientas hechas a medida para desarrollar aplicaciones para el Internet de Futuro. Ya se trate de plug-ins personalizados para Eclipse, suites de pruebas de software o directrices y mejoras prácticas. A continuación se resumirán las herramientas que hay disponibles en el Catálogo.

FI-WARE NGSI10 Test Server

Es una pieza de software creada para el propósito de realizar pruebas de software. No es un GE en sí mismo, sino lo que permite probar el software que el usuario utiliza para interactuar con los GE del Internet de las Cosas que hacen uso de FI-WARE NGSI10.

FIA Project Management Plugin

Permite a los usuarios de FI-CoDE crear aplicaciones del Internet del Futuro que utilizan instancias de GE de FI-WARE.

Este plugin permite crear proyectos basados en Java, navegar y seleccionar instancias disponibles en el catálogo, que están asociadas a la configuración de proyectos FIA.

FusionForge Connector

Es un plugin de Eclipse para utilizar Mylyn. Proporciona una interfaz nativa de eclipse que permita a los desarrolladores crear y actualizar tickets y tareas sin necesidad de abandonar el IDE.

PROSA

Una herramienta para monitorización continua y visualización del tiempo de respuesta de servicios. Puede ser configurada para invocar servicios compuestos por pruebas en línea, de modo que permite reunir tiempos de respuesta adicionales cuando sea necesario.

REST Client Generator

Es un plugin de Eclipse que arrancando desde un archivo WADL (formal description of the RESTful services) es capaz de crear el código fuente java que rodea la interacción entre la aplicación y el servicio remoto.

SoPeCo

Siglas de *Software Performance Cockpit,* es un marco de trabajo para evaluaciones sistemáticas del rendimiento del software, basado en medidas sistemáticas, métodos estadísticos y aprendizaje automático.

Trace Analyzer

Es un espacio de trabajo basado en Eclipse para agrupar, analizar y visualizar datos de rendimiento. Su objetivo es ayudar al usuario a encontrar cuellos de botella en el funcionamiento. Utilizando datos de varias fuentes y diferentes tiempos permite la visualización gráfica de las ejecuciones de los programas así como listas de detalle.

Unit Functional Testing Framework

Es un proyecto java que contiene un set de librerías y una estructura de paquetes que facilita y conduce la implementación de pruebas para validar el comportamiento de una aplicación que expone servicios web con RESTfull.

3.3 Arduino UNO

3.3.1 Vista Conjunta

Ilustración 3.11. Vista Conjunta Arduino UNO

La placa de Arduino UNO está basada en el micro-controlador ATMEGA328. Tiene 14 pines digitales de entrada/salida (de los cuales 6 se pueden utilizar como salidas PWM), 6 entradas analógicas, una conexión USB utilizada para cargar los programas en la placa y para la comunicación serie entre la placa y el ordenador (puede utilizarse como alimentación a la placa), un conector Jack DC de alimentación externa, un programador serie en circuito "In-circuit Serial Programmer" o "ICSP" y un botón de reinicio.

Incluye 2 pines SDA (línea de datos) y SCL (línea de reloj) localizados cerca del pin AREF (terminal de referencia analógica). Además, internamente incluye un selector de alimentación entre el USB y el conector de alimentación externa. El pin IOREF permite que los Shields se adapten al voltaje empleado por la placa y el pin reset es utilizado para añadir un botón de reset a los Shields.

3.3.2 Resumen

Característica	**Descripción**
Micro-controlador	ATMEGA328
Voltaje Entrada (Recomendado)	7 – 12 V
Pines E/S Digitales	14 (6 proporcionan salida PWM)
Pines Entrada Analógica	6
Corriente Pines E/S	40 mA
Corriente Pin 3,3 V	50 mA
Memoria Flash	32 KB (0,5 KB para el bootloader)
SRAM	2 KB
EEPROM	1 KB
Velocidad Reloj	16 MHz

Tabla 3.3. Resumen de las características de la placa Arduino.

3.3.3 Memoria

El ATMEGA328 tiene 32 KB de Memoria Flash para almacenar código (de los cuales 0,5 KB se usan para el bootloader). Tiene 2 KB de SRAM y 1 KB de EEPROM (que puede ser leída y escrita con la librería EEPROM).

3.3.4 Funciones Pines E/S

Cada uno de los 14 pines digitales de Arduino UNO puede ser utilizados como entrada o salida, utilizando las funciones "digitalRead()", "digitalWrite()" y "pinMode()". Funcionan a 5 V. Cada pin puede proporcionar o recibir un máximo de 40 mA y tiene una resistencia pull-up interna (desconectada por defecto) de 20 a 50 kΩ. Además, algunos pines tienen funciones especiales:

Serie: Pines 0 (RX) y 1 (TX). Se utiliza para recibir (RX) y de transmitir (TX) datos en serie usando niveles TTL (Transistor-Transistor Logic).

Interrupciones Externas: Pines 2 y 3. Estos pines pueden ser configurados para activar una interrupción en un valor bajo, un flanco ascendente o descendente, o un cambio de valor.

PWM: Pines 3, 5, 6, 9, 10 y 11. Proporcionan salida PWM (Pulse Width Modulation) de 8 bits.

SPI: Pines 10 (SS), 11 (MOSI), 12 (MISO) y 13 (SCK). Estos pines soportan la comunicación SPI usando la librería SPI.

Arduino UNO tiene 6 entradas analógicas, A0 – A5, cada una de las cuales proporcionan 10 bits de resolución, esto es, 1024 valores diferentes.

TWI: Pines A4 o SDA y pines A5 o SCL. Soporta comunicación TWI usando la librería Wire.

AREF: Voltaje de referencia para entradas analógicas.

Reset: Pin para resetear el micro-controlador, normalmente se utiliza para agregar un botón de reinicio para los Shields.

3.3.5 Comunicación

Arduino UNO permite comunicarse con un ordenador, otro Arduino, u otros micro-controladores. El ATMEGA328 provee comunicación serie UART TTL (5 V), la cual está

disponible en los pines digitales 0 (Rx) y 1 (Tx). Un FTDI FT232RL en la placa canaliza esta comunicación serie al USB y los drivers FTDI (incluidos con el software Arduino) proporcionan un puerto de comunicación virtual al software del ordenador. El software Arduino incluye un monitor serie que permite intercambiar datos con la placa Arduino.

El ATMEGA328 también soporta comunicación I^2C (TWI) y SPI.

3.3.6 Programación

Arduino UNO puede ser programado con el software Arduino vía USB. El propio software lleva consigo los drivers necesarios para emular el puerto serie.

El ATMEGA328 de Arduino UNO viene con un bootloader pregrabado que permite cargar nuevo código sin usar un programador hardware externo. Se comunica usando el protocolo original STK500. Es algo poco habitual en otros tipos de micro-controladores, que generalmente requieren un programador externo.

También es posible eliminar el bootloader y programar el ATMEGA328 a través del conector ICSP (In-Circuit Serial Programming).

3.3.7 Protección de Sobrecarga del USB

Arduino UNO tiene un fusible que protege los puertos USB del ordenador de cortes y sobrecargas. Aunque la mayoría de los ordenadores proporcionan su propia protección interna, el fusible proporciona una protección extra. Si se aplican más de 500 mA al puerto USB, el fusible automáticamente interrumpirá la conexión hasta que el corte o la sobrecarga sean eliminados.

3.3.8 Alimentación

Arduino UNO puede ser alimentado a través de la conexión USB o con una fuente de alimentación externa, realizándose la selección de manera automática.

La placa puede funcionar con un suministro externo de 6 - 20 V. Si se proporcionan menos de 7 V, el pin de 5 V puede suministrar menos de 5 V y el micro-controlador puede no funcionar correctamente. Por otro lado, si se alimenta con más de 12 V, el regulador de voltaje se puede sobrecalentar y dañar la placa. El rango recomendado es de 7 a 12 V.

Vin: Se utiliza para la alimentación de la placa. El voltaje suministrado debe estar comprendido entre 7 y 12 V. En caso de alimentar a la placa mediante el conector de alimentación externo se puede utilizar este pin como salida.

5 V: Salida a 5 V. No se recomienda alimentar la placa a través de este pin, ya que no pasa por la protección ante sobrecarga y puede dañar la placa.

3,3 V: Salida a 3,3 V, 50 mA de corriente máxima.

GND: Pines de tierra.

IOREF: Este pin de la placa Arduino UNO proporciona la referencia de tensión con la que opera el micro-controlador a los Shields.

3.3.9 Software

El entorno de código abierto Arduino hace fácil escribir código y programarlo en la placa. Es un entorno multiplataforma, pudiendo ser utilizado en Windows, Macintosh y Linux.

El entorno de desarrollo (IDE, Integrated Development Environment) está escrito en Java y basado en Processing, avr-gcc y otros programas también de código abierto.

Es un entorno fácil de usar, es simple y directo, dando flexibilidad tanto a usuarios principiantes como avanzados.

3.3.9.1 Librerías

Arduino ofrece una serie de librerías "estándar" basadas en C/C++ que se pueden importar al Sketck. Las bibliotecas "estándar", principalmente, son las siguientes:

- EEPROM: Para leer y escribir en memorias "permanentes".
- Ethernet: Para conectar a Internet usando el Ethernet Shield.
- Firmdata: Para comunicarse con aplicaciones en la computadora usando un protocolo estándar Serial.
- LiquidCrystal: Para controlar Displays de cristal líquido (LCD).
- Servo: Para controlar servomotores.
- SoftwareSerial: Para la comunicación serial de cualquier pin digital.
- Stepper: Para controlar motores paso a paso.
- Wire: Interfaz de dos cables (TWI/I^2C), para enviar y recibir datos a través de una red de dispositivos y sensores.

Además de estas librerías estándar, es posible encontrar otras muchas en la red desarrolladas por la comunidad de usuarios.

3.3.9.2 Diseño Sketch

El diseño de un Sketch para Arduino no requiere de avanzados conocimientos en electrónica y en programación. Así, por ejemplo, para configurar una línea digital como salida se puede utilizar la siguiente función *pinMode(13, OUTPUT)*. De mismo modo, para escribir un valor alto o bajo en dicho pin se utiliza la siguiente función *digitalWrite(13, HIGH or LOW)*. Además, para realizar tareas más complejas, tales como gestionar una comunicación por UART, se pueden utilizar las siguientes funciones *Serial.begin(speed)*, *Serial.read()*, *Serial.available()*, etc.

La estructura básica del lenguaje de programación Arduino es bastante simple y se organiza en al menos dos partes o funciones que encierran bloques de declaraciones.

```
void setup() {
        Statements;
}
void loop(){
        Statements;
}
```

Ambas funciones son requeridas para que el programa funcione.

- setup(): La función "setup" debería contener la declaración de cualquier variable al comienzo del programa. Es la primera función a ejecutar en el programa. Concretamente, es ejecutada una vez y usada por ejemplo para inicializar las comunicaciones serie.

- loop(): La función "loop" se ejecuta a continuación e incluye el código que se ejecuta continuamente leyendo entradas, activando salidas, etc. Esta función es el núcleo de todos los programas Arduino y realiza la mayor parte del procesamiento.

3.3.9.3 Entorno de Desarrollo

Como ya se ha mencionado anteriormente, el entorno de desarrollo de Arduino es gratuito y no requiere instalación (http://arduino.cc/en/Main/Software).

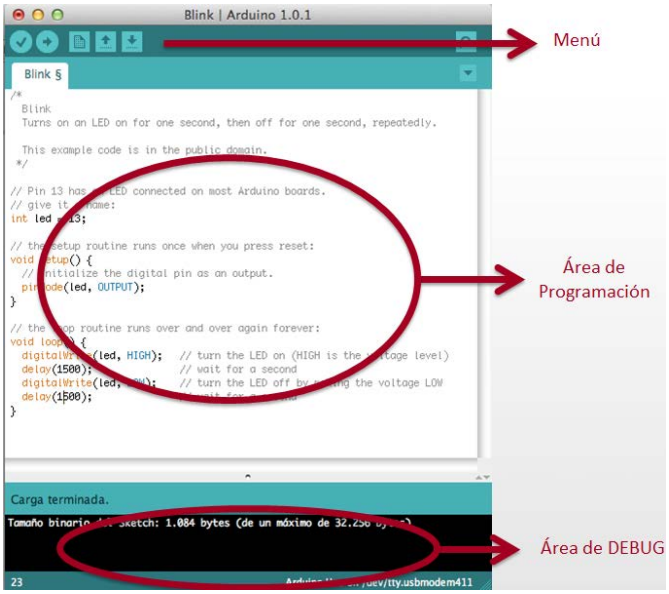

Ilustración 3.12. Vista entorno de Desarrollo Arduino

En el siguiente enlace se encuentra toda la información necesaria para la instalación del software en la plataforma Windows, Mac o Linux (http://arduino.cc/es/Guide/HomePage).

Ilustración 3.13. Vista menú Entorno de Desarrollo

El software desarrollado con Arduino se conoce como sketches. Los sketches se escriben con un editor de texto y son guardados con la extensión ".ino".

3.3.10 Shields

Las Shields son placas que pueden ser conectadas encima de la placa Arduino extendiendo sus capacidades. Para ello, los pines de sus puertos guardan una disposición de compatibilidad.

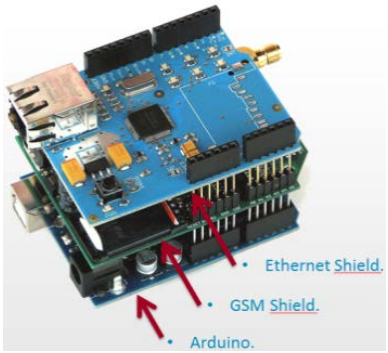

Ilustración 3.14. Vista Arduino con Shields GSM y Ethernet

Las diferentes Shields siguen la misma filosofía que el conjunto original: son fáciles de montar y económicas de producir. Existe además una gran variedad de Shields con diversa funcionalidad: control de motores, comunicaciones, prototipado rápido, etc.

3.3.10.1 Wifi shield

Arduino WiFi shield conecta Arduino a Internet de forma inalámbrica. Sus principales características se enumeran a continuación:

- Opera con 5 V que aporta la propia placa de Arduino
- Conexión vía redes 802.11b/g
- Encriptación soportada: WEP y WPA2 Personal
- Conexión con Arduino mediante el puerto SPI
- Slot micro-SD
- Conexión ICSP
- Mini-USB para actualización del firmware del dispositivo

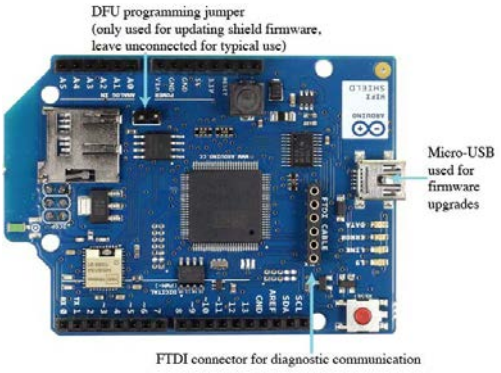

Ilustración 3.15. Vista Wifi shield con anotaciones

El dispositivo permite la conexión con Internet mediante especificaciones 802.11. Está basado en sistema HDH104 Wireless LAN 802.11 b/g. Un Atmega 32UC3 permite usar comunicaciones tanto TCP como UDP.

Existe un slot micro-SD *onboard* que puede ser usado para almacenar archivos en la red y que es compatible con Arduino Uno y Mega.

Este shield tiene su propia librería con diferentes Sketches de ejemplo para la utilización tanto de la conexión a Internet como lectura y escritura en la tarjeta SD. Cuando se trabaja con ésta última, el chip select (SS) se encuentra en el pin 4 por lo que éste deja de ser funcional.

Arduino se comunica con Wifi shield a través del puerto SPI. Este bus utiliza los pines digitales 11,12 y 13 del Arduino Uno, por lo que también dejan de ser funcionales. Además, ocurre lo mismo con el pin 10, ya que es utilizado para seleccionar el HDG104. En la Ilustración 3.16 se muestra un esquema resumen de los pines que dejan de ser operativos.

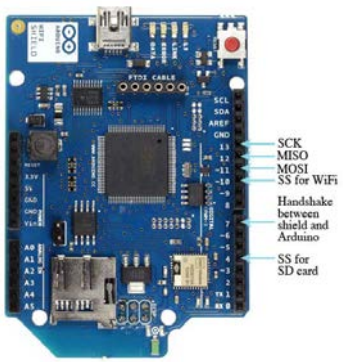

Ilustración 3.16. Vista de los pines que dejan de ser operativos

Es de interés saber que cuando alguna de las librerías (Wifi o SD) no se utilizan, es posible recuperar la funcionalidad de los pines que corresponden con esa conexión. Sin embargo, no se hace de forma automática, sino que se deben deseleccionar explícitamente.

El conector Mini-USB se utiliza para actualizar el Atmega 32U, utilizando el protocolo Atmel DFU. Para ello el jumper señalado en la Ilustración 3.15 debe ser conectado.

El shield contiene un determinado número de LEDs destinados a informar al usuario sobre el estado de funcionamiento.

- L9 (amarillo): ligado al pin 9.
- LINK (verde): indica si se ha realizado la conexión con la red.
- ERROR (rojo): indica cuando hay un error en la comunicación.
- DATA (azul): indica cuando se están transmitiendo o recibiendo datos.

3.4 Raspberry Pi modelo B

3.4.1 Vista conjunta

La Raspberry Pi modelo B (Ilustración 3.17), como ya se expuso en el capítulo anterior, es un mini-PC que cuenta con un procesador ARM1176JZFS de 700 MHz montado en un socket BCM2835, un núcleo gráfico de procesos con cuatro GPUs y 512 GB de RAM. Dispone de dos conexiones USB, puerto Ethernet, Jack de audio de 3.5 mm y HDMI, y es capaz de reproducir vídeo de calidad BluRay utilizando H.264 a 40 Mbits/s, gracias a la aceleración 3D aportada por las librerías OpenGL ES2.0 y OpenVG.

Ilustración 3.17. Vista de la Raspberry Pi modelo B

Su capacidad de cálculo, indispensable en la mayoría de proyectos, es de 24 GFLOPS de computación de uso general. En comparación con un PC de sobremesa, su rendimiento puede asemejarse a un Pentium 2 con 3000 MHz, sólo que con mucho mejor rendimiento gráfico.

3.4.2 Resumen.

Características	**Descripción**
Procesador	ARM1176JZFS
Voltaje Funcionamiento	5 V
Voltaje Entrada (Recomendado)	4,75 – 5,25 V
Voltaje Entrada (Límites)	5,25 V
Pines E/S Digitales	17 (1 PWM proporcionado por jack)
Pines SPI	5
Pines I²C	2
Pines UART	2
Pines Entrada Analógica	17
Corriente Pines E/S	16 mA por pin (56 mA en total)
Corriente Pin 5/3,3 V	300 mA
Memoria Flash	Tarjeta SD (Hasta 32 GB)
Velocidad Reloj	700 MHz

Tabla 3.4. Resumen de las características de la placa Raspberry Pi.

3.4.3 Memoria

Raspberry Pi B cuenta con 512 MB de memoria RAM, hasta 32 GB de memoria flash con tarjeta SD y para almacenamiento, siempre es posible conectar por USB dispositivos de mayor capacidad.

Es importante tener en cuenta que como el software se instala en la tarjeta SD, influye mucho la clase de dicha tarjeta, ya que ésta a su vez determina la velocidad de transferencia que se traduce finalmente en un mejor rendimiento de la Raspberry Pi.

Cabe mencionar que no todas las tarjetas SD existentes en el mercado son compatibles con este dispositivo. Es por ello que existen varios listados donde comprobar que un determinado modelo y tamaño es compatible con el sistema. Un ejemplo de este tipo de listas es: http://elinux.org/RPi_SD_cards.

3.4.4 GPIO

El SBC dispone de varias GPIOs (*General Purpose Input/Output)* de Raspberry Pi (ver Ilustración 3.18). De todos los existentes, 17 son realmente GPIO, el resto son pines de salida de voltaje y GND.

Ilustración 3.18. Vista del GPIO de Rasbperry Pi modelo B

En la Ilustración 3.19 se muestra la configuración de cada pin mediante un código de colores. Los pines de tipo *GPIO* mostrados se emplean en la entrada y salida digital.

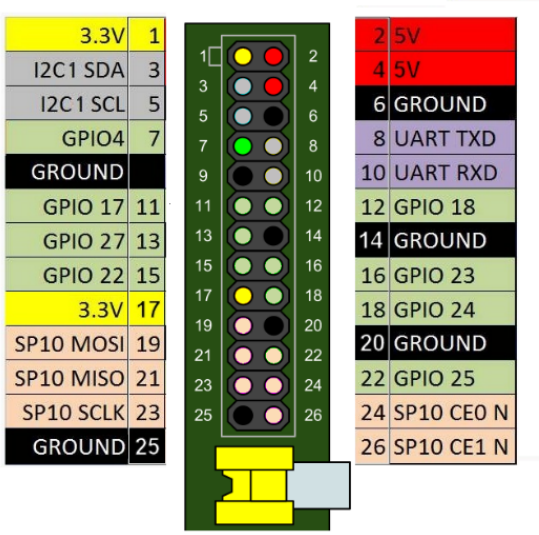

Ilustración 3.19. Configuración de los pines del GPIO

3.4.5 Riesgos de sobrecarga

Raspberry Pi posee diferentes entradas y salidas al sistema y es conveniente conocer cuáles son los límites que se pueden alcanzar en su funcionamiento.

- La alimentación del sistema Raspberry se realiza a través de micro-USB, de forma que su tensión de alimentación debe ser de 5 V, admitiendo una tolerancia de ±0,25V. Si se sobrepasa este valor el sistema viene preparado con un fusible de protección de 1,1 A.

- Los USB disponibles permiten la conexión de otros dispositivos externos, sin embargo, hay que controlar la intensidad que estará aportando dado Raspberry Pi modelo B ya que no está limitada. Así, la intensidad que se puede demandar a los USB depende de la intensidad con la que se alimenta Raspberry Pi. Esto hace que el límite teórico sea de 1,1 A, que es el máximo permitido por el fusible menos la intensidad demandada por la placa. Hay que tener en cuenta que dispositivos de gran impedancia pueden hacer que se reinicie el dispositivo por caída de tensión al conectarlos.

- El conector GPIO posee unos pines que de forma continuada aportan 5 V y 3,3 V. Los pines de 5 V permiten un consumo acorde con la alimentación del dispositivo (al igual que las salidas USB). En el caso de los pines de 3,3 V no debe conectarse nada que haga que la intensidad supere 50 mA. En general como máximo, con una fuente normal de alimentación, no se recomienda superar los 300 mA.

- Los pines I/O digitales no deben tener conectados nada que consuma más de 16 mA de intensidad con un total de 50 mA entre todos los pines.

Como curiosidad, cabe destacar que en los USB del modelo A o Rev1 estaba limitada la intensidad de salida mediante fusibles a 140 mA. Estos fusibles resultan tener una resistencia de 5 Ω, ocasionando que la tensión en el USB sea de 4,5V, traduciéndose en un mal funcionamiento de algunos dispositivos externos. Por ello, se decidió eliminar esta protección en el modelo B.

3.4.6 Programación

Dado que la mayoría de sistemas operativos están basados en Linux, la programación de la Raspberry Pi se realiza mediante cualquier programa válido para Linux siempre y cuando exista el compilador adecuado para arquitectura ARM.

De forma general, la programación trae consigo la interacción con GPIO y ésta se puede realizar de varias formas, aquí se muestran algunas:

- Con la línea de comandos. El acceso a los pines del GPIO se realiza con comandos de la propia línea de Linux en tres pasos. Primero se selecciona el pin a utilizar mediante el comando *echo [nº_pin] > /sys/class/gpio/export*. En segundo lugar se señala si el pin es de salida o entrada con el comando *echo [in/out] > /sys/class/gpio/gpio17/direction*. Por último, se señala el valor del pin. En el caso de que se haya seleccionado que es de salida con *echo 0 > /sys/class/gpio/gpio17*/value o se recoge si se seleccionó de entrada.

- Utilizando Python. En este caso se emplean funciones específicas de una librería para python llamada RPi.GPIO. Para seleccionar que un pin será de entrada o salida se utilzará la función *GPIO.setup(17, GPIO.[IN/OUT])*. Para seleccionar un valor o leerlo se utilizarán respectivamente *GPIO.output() y GPIO.input()*.

- Mediante una herramienta de hardware de Matlab. Matlab posee una herramienta que permite la interacción con el GPIO mediante sencillas funciones. Para señalar que un pin será de entrada o salida se utiliza *configureDigitalPin(mypi,4,'input/output')* y para leer o escribir un valor se usan las funciones *readDigitalPin()* y *writeDigitalPin()* .

3.4.7 Software

Como ya se mencionó en el capítulo anterior, hay disponibles varios sistemas operativos para Raspberry Pi, sin embargo, aquí se van a detallar las características de *Raspbian*, ya que es el software que se va emplear en el presente proyecto. Se ha seleccionado dicho sistema operativo por su uso extendido y su gran versatilidad.

Raspbian es un sistema operativo gratuito basado en Debian Wheezy y optimizado para el hardware de Raspberry Pi. Sin embargo, aporta mucho más que un sistema operativo puro, ya que cuenta con más de 35000 paquetes de software pre-compilado, que hace más sencilla la instalación en el dispositivo. Desde su primera versión en junio de 2012, Raspbian está en continuo desarrollo aumentando el citado número de paquetes y haciéndose cada vez más estable.

En diciembre de 2012, junto a la versión 2012-12-16-wheezy-raspbian de Raspbian, se lanzó la tienda de aplicaciones *Pi Store*, que en el momento de salida incluía aplicaciones como *LibreOffice* o *Asterisk*. En esta plataforma se puede poner a disposición de todos los

usuarios de Raspbian contenidos gratuitos o de pago, como archivos binarios, código python, imágenes, audio o vídeo.

La distribución usa *LXDE* como escritorio y *Midori* como navegador web. Además, contiene herramientas de desarrollo como *IDLE* para el lenguaje de programación *Python* o *Scratch*.

Destaca también el menú *raspi-config,* que permite configurar el sistema operativo sin tener que modificar archivos de configuración manualmente. Entre sus funciones, permite expandir la partición root para que ocupe toda la tarjeta de memoria, configurar el teclado, aplicar overclock, etc.

Al igual que ocurre con cualquier versión de Linux, posee dos posibles interfaces de usuario. Una línea de comandos y una interfaz gráfica accesible mediante el comando *startx*. En la Ilustración 3.20 aparece la interfaz gráfica que por defecto muestra Raspbian tras su instalación.

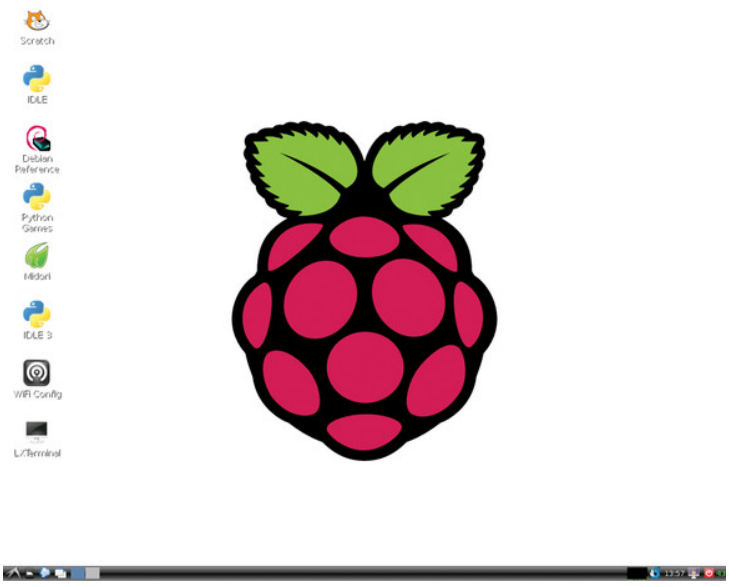

Ilustración 3.20. Vista interfaz gráfica de Raspbian

Toda la información, novedades, ayudas para instalación, foros y mucho más se puede encontrar en la web de Raspbian: http://raspbian.org/

3.4.7.1 Instalación de Raspbian

La instalación de Raspbian se realiza en una tarjeta SD. Para obtener la imagen de la distribución existen varias URL disponibles, sin embargo es recomendable utilizar la oficial: http://www.raspberrypi.org/downloads/

En primer lugar se debe cargar la imagen descargada en la tarjeta SD. Para ello se puede utilizar algún programa disponible. Uno gratuito y open source disponible es *Win32 Disk Imager*. Una vez conectada la SD al PC, se abre el programa y se indica que se va a copiar en la tarjeta y se selecciona el directorio con la imagen de Raspbian descargada. Este proceso se puede ver en la Ilustración 3.21.

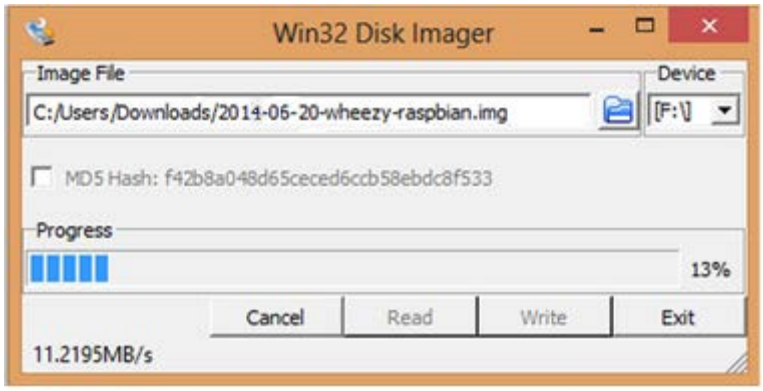

Ilustración 3.21. Preparación de la tarjeta SD para instalación de Raspbian

Una vez finalizado el proceso, se introduce la tarjeta en la Raspberry Pi y se inicia automáticamente el proceso de instalación. Una vez finalizado, se pueden realizar algunos ajustes, tales como seleccionar idioma, el idioma del teclado y el tamaño de partición se quiere asignar al SO, entre otros (ver Ilustración 3.22), gracias al ya citado menú Raspi-config.

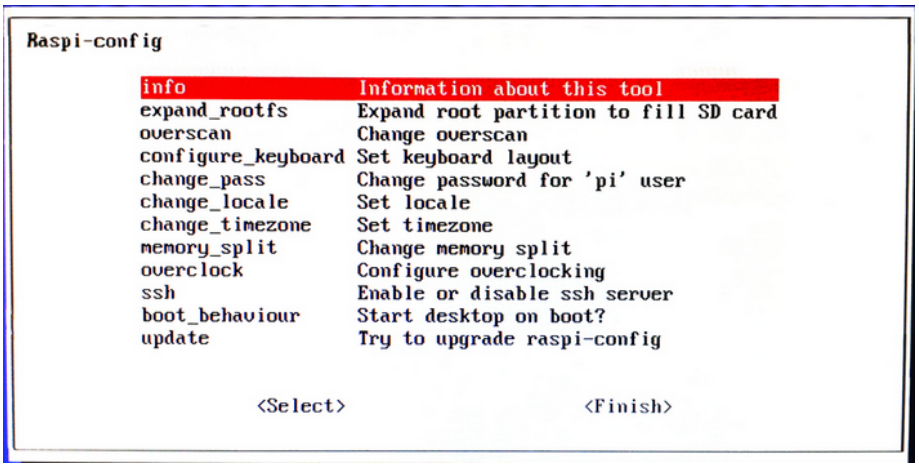

Ilustración 3.22. Ajustes disponibles tras la instalación

3.5 Sensores

A lo largo del presente proyecto se desarrollará un caso práctico en el que se emplearán una serie de sensores de temperatura, ultrasonido y PIR. En este apartado se describirá cada uno de ellos.

3.5.1 Sensor de temperatura y humedad relativa SHT11

El SHT11 (ver Ilustración 3.22) es un circuito integrado para la medición de temperatura y humedad relativa de elevada precisión que entrega una salida digital. Este componente se puede conectar mediante I^2C gracias a sus cuatro canales: datos (DATA), señal de reloj (SCK), tierra (GND) y alimentación (VDD). Esto facilitará en gran medida la conexión con Arduino y Raspberry Pi.

El circuito necesita estar alimentado con valores de voltaje de entre 2,4 y 5,5 V. Además su consumo es bajo, en torno a los 30 μW.

Ilustración 3.22. Vista sensor SHT11

Este dispositivo cuenta con las especificaciones mostradas en la tabla 3.5

Parámetro	min	típico	max	Unidades
Resolución	0,04	0,01	0,01	ºC
	12	14	14	bits
Precisión		± 0,4		ºC
Repetibilidad		± 0,1		ºC
Rango de operación	-40		123,8	ºC
Tiempo respuesta	5		30	s

Tabla 3.5. Hoja de especificaciones SHT11

En la comunicación con el sensor se utilizan comandos digitales, que es conveniente conocer (ver Tabla 3.6), aunque en la mayoría de los caso existen librerías disponibles para los diferentes dispositivos empotrados.

Comando	Código
Reservado	0101x-1110x
Medir Temperatura	00011
Medir Humedad	00101
Leer registro de est. Int.	00111
Escribir registro	00110
Reset	11101

Tabla 3.6. Lista de comandos digitales para SHT11

3.5.2 Sensor de temperatura TMP36GZ

El sensor TMP36GZ (ver Ilustración 3.23) es un sensor de temperatura de bajo voltaje y de tipo analógico que envía una señal eléctrica linealmente proporcional a una escala centígrada. No necesita calibración externa para un correcto funcionamiento con precisiones de ±1 ºC a los 25 ºC y de ±2 ºC en valores límites de entre -40 ºC y 125 ºC.

Ilustración 3.23. Vista del sensor analógico TMP36GZ

Este dispositivo se alimenta con valores de voltaje de entre 2,7 y 5,5 V, aportando 250 mV de salida a 25ºC.

En la tabla 3.7 se muestran las características generales de este sensor.

Parametro	Condiciones	Min	Típico	Máx	Unidad
Precisión	25 ºC		±1		ºC
	Límite		±2		ºC
Apagado	Alto voltaje	1,8			mV
	Bajo voltaje			400	mV
Voltaje salida	25 ºC		250		mV

Tabla 3.7. Características generales TMP36GZ

3.5.3 Sensor de ultrasonidos SRF02

El sensor de distancias por ultrasonidos SRF02 (ver Ilustración 3.24) es un sensor de pequeño tamaño y mínimo consumo. Posee un único transductor, lo que lo hace mucho más simple que otros existentes en el mercado.

Destaca por tener interfaz serie e interfaz I^2C. La interfaz serie tiene un formato estándar de 9600 baudios, un bit de comienzo, ocho de datos y un bit de parada. El nivel de tensión es TTL, lo que permite conectarlo a cualquier microcontrolador del mercado. En ambos modos el rango de medidas es de 15 cm a 600 cm, con una precisión de 1 cm. El ciclo de medida es de 45 ms.

Cada sensor tiene su propia dirección interna, aunque esta se puede cambiar de forma que se pueden tener hasta 16 módulos SRF02 en el mismo bus, ya sea serie o I2C. Las medidas pueden ser en centímetros, pulgadas o microsegundos. La alimentación es de 5 V y el consumo medio de 4 mA. Puede conectarse directamente a un PC por USB utilizando el circuito interfaz S310425.

Cuenta con cinco canales: datos, señal de reloj, tierra, alimentación y modo (para conmutar de I^2C a Serie).

Ilustración 3.24. Vista del sensor de ultrasonido SRF02

Como en cualquier sensor digital, éste cuenta con una serie de comandos que permiten comunicarse con él. Dichos parámetros se muestran en la Tabla 3.8.

Comandos		Descripción
Decimal	**Hexadecimal**	
80	0x50	Iniciar una nueva medición real. Resultado en pulgadas
81	0x51	Iniciar una nueva medición real. Resultado en centímetros
82	0x52	Iniciar una nueva medición real en microsegundos
86	0x56	Iniciar una nueva medida falsa. Resultado en pulgadas
87	0x57	Iniciar una nueva medida falsa. Resultado en centímetros
88	0x58	Iniciar una nueva medida falsa. Resultado en microsegundos
92	0x5C	Transmite una ráfaga de 8 ciclos de 40khz. No calcula.
96	0x60	Fuerza un reinicio del sonar mediante un ciclo de autoajuste.
160	0xA0	1º comando de la secuencia para cambiar la dirección I2C
165	0xA5	3º comando de la secuencia para cambiar la dirección I2C
170	0xAA	2º comando de la secuencia para cambiar la dirección I2C

Tabla 3.8. Comandos para el sensor SFR02.

3.5.4 Sensor de ultrasonidos HC-SR04

El sensor de distancias por ultrasonidos HC-SR04 (ver Ilustración 3.25) es un sensor de mediano tamaño. Posee dos transductores, lo que lo hace mucho más preciso que el SRF02 descrito.

Tiene interfaz con cuatro pines característicos: GND, VDD, TRIGGER y ECHO. Esto se debe a que este sensor tiene un funcionamiento en tres pasos:

- Se emplea una entrada a escalón de al menos 10 μs en la conexión *Trig* para comenzar la medición.

- El módulo automáticamente envía una señal de 40 kHz y detecta cuando vuelve dicho pulso al receptor.

- Si el tiempo de retorno de la señal es el tiempo que la salida *echo* mantiene un nivel alto de tensión. Dicho tiempo de ida y vuelta será el utilizado para calcular la distancia.

La distancia será:

$$Distancia = \frac{(tiempo\ en\ nivel\ alto) * (velocidad\ del\ sonido\ (340m/s))}{2}$$

El rango de medidas es de 2 cm a 400 cm, con una precisión de 1 mm. Su ciclo de medida es de 60 ms.

La alimentación es de 5 V, consumo medio de 15 mA y su frecuencia de trabajo es de 45 Hz

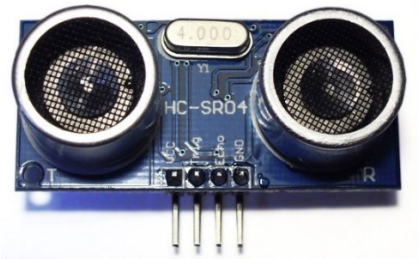

Ilustración 3.25. Vista del sensor HC-SR04

Dependiendo de su ángulo de inclinación éste tendrá un rendimiento muy distinto. En la Ilustración 3.26 se muestra un gráfico mostrando dicho rendimiento frente al ángulo de medida. Se demuestra que debe ser menor de 30º para una buena medida.

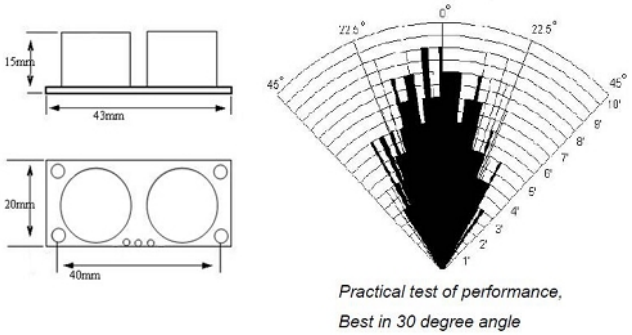

Ilustración 3.26. Rendimiento del sensor HC-SR04 frente al ángulo de medida

3.5.5 Sensor Infrarrojo Pasivo (PIR) HC-SR501

El HC-SR501 (ver Ilustración 3.27) sensor capaz de detectar movimiento mediante infrarrojos a 7 metros de distancia (decreciendo con la temperatura) y dentro de un ángulo de 120º.

Ilustración 3.27. Vista del sensor PIR HC-SR501

Como se puede apreciar en la Ilustración 3.27 posee dos resistencias variables regulables, Ch1 y RL2. Variando el valor de Ch1 es posible establecer el tiempo que se mantiene activa la salida del sensor tras detectar movimiento, limitado a un mínimo de 3 s, salvo que se sustituya la resistencia. Cambiando el valor de RL2 se establece la distancia de detección que puede variar entre 3-7 m.

La posibilidad de mantener activa la salida del módulo durante un tiempo determinado permite poder usarlo directamente para muchas aplicaciones sin necesidad de usar un microcontrolador.

Cuenta además con dos formas de detección o disparo seleccionables mediante un jumper:

- Disparador no repetible: Cuando sensor envía un nivel de tensión alto (detecta actividad), se acaba el intervalo y automáticamente el valor de tensión cambia a bajo.

- Disparador repetible: El valor de tensión suministrado por el sensor es alto durante el tiempo de retardo (delay), mientras haya actividad humana en rango permanecerá alto hasta que deje de haber actividad y transcurra el tiempo de muestreo.

En la Tabla 3.9 se muestran las principales características técnicas de este dispositivo.

Parámetro	Valor/Rango
Voltaje	5V-20V
Consumo	65 mA
Consumo	3,3V, 0V
Tiempo de bloqueo	0,2 s
Temperatura	-15 a 70 ºC
Rango sensibilidad	120º en 7m

Tabla 3.9. Características del sensor PIR HC-SR501

Capítulo 4

Caso de estudio. Descripción del hardware y software desarrollado.

4.1 Introducción.

Tras la descripción realizada en los capítulos anteriores, ya es posible llevar a cabo la implementación de un caso de estudio concreto que muestre el desempeño de la plataforma FI-WARE. En primer lugar se detallará el caso de estudio de forma general, comentando su funcionamiento y objetivos. Posteriormente, se describirá el hardware y software de forma más específica.

El estudio, en líneas generales, consiste en la implementación de un sistema de parking inteligente, que hace más eficiente la tarea de uno convencional.

El parking inteligente surge como aplicación de las nuevas tecnologías de Smart Cities para la mejora de la calidad del servicio a los usuarios. Es de gran utilidad en grandes ciudades donde escasean las zonas de aparcamiento. Con el sistema propuesto será posible visualizar el número de plazas libres, cuales son dichas plazas (visualizándolas en un mapa del emplazamiento), visualizar algunos parámetros climáticos de cada plaza e incluso reservar plazas libres, todo ello antes de salir de casa y en tiempo real.

Cuanto mayor sea el número de parkings que utilicen este sistema, mayor será la utilidad, ya que será posible decidir a qué lugar acudir sin tener que buscar en el momento, con evidentes pérdidas de tiempo y dinero.

El gestor del parking contará también con grandes herramientas de análisis, gracias al histórico de datos del que puede disponer y de donde se podrán obtener información sobre la ocupación de cada plaza, temperaturas a lo largo del día o el tránsito de vehículos a lo largo del tiempo, permitiendo la mejora y evolución del sistema.

4.2 Descripción del caso de estudio.

El parking seleccionado para llevar a cabo el estudio es uno rectangular con 14 plazas de aparcamiento. En el sistema están distribuidos 16 dispositivos: 2 dispositivos en la entrada y salida del parking y uno en cada una de las plazas. En la Ilustración 4.1 se muestra el esquema del sistema, donde se muestra la ubicación de los dispositivos de entrada y salida (E y S respectivamente) y la de los demás correspondientes a cada aparcamiento.

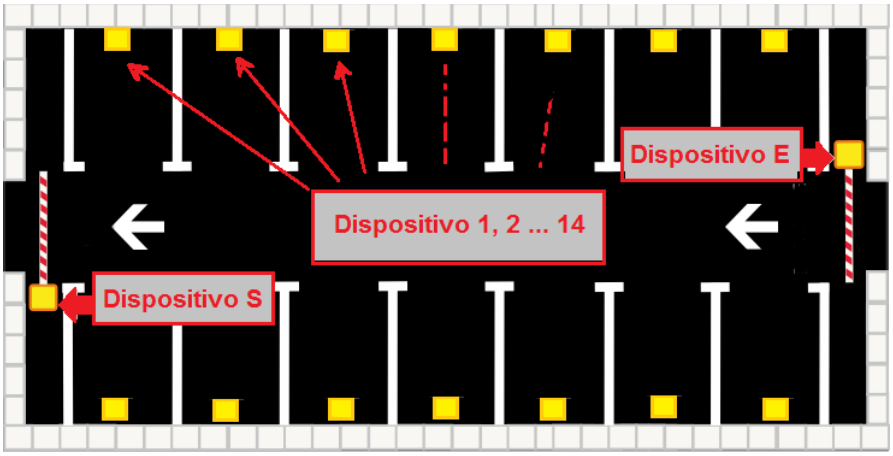

Ilustración 4.1. Vista del parking del caso de estudio y sus dispositivos.

Dado que en todas las plazas de aparcamiento el hardware y software es idéntico, carece de sentido hablar de 14 dispositivos, y dado que se pretende implementar dos variedades de dispositivos para las plazas, el caso de estudio se reducirá al desarrollo de 2 dispositivos para los aparcamientos y otros 2 para entrada y salida del emplazamiento, siendo así extrapolable el estudio al caso real.

De esta forma, los dispositivos a implementar serán en definitiva los enumerados a continuación:

- Dispositivo 1. Ubicado en la entrada del parking (Dispositivo E en la Ilustración 4.1). Basado en Arduino. Señalizará si hay plazas disponibles o no en el parking mediante LEDs verde y rojo, así como detectar los vehículos que entran.

- Dispositivo 2. Ubicado en la salida (Dispositivo S en la Ilustración 4.1). Basado en Arduino. Detectará los vehículos que abandonan el parking.

- Dispositivo 3. Primera variedad de los dispositivos ubicados en plaza de aparcamiento. Basado en Arduino. Dispone de LEDs para señalizar si está ocupada o no la plaza y medirá variables climáticas.

- Dispositivo 4. Segunda variedad de los dispositivos ubicados en plaza de aparcamiento. Basado en Raspberry Pi. Dispone de LEDs para señalizar si está ocupada o no la plaza, medirá condiciones climáticas y tendrá la posibilidad de estar reservado.

En lo que concierne a FI-WARE, será necesario crear una instancia con diferentes entidades para el correcto funcionamiento del sistema.

En la sección de Cloud de FI-Lab se encuentra instanciada una máquina virtual llamada *UPCT*, con la versión 0.14.0 de *orion*, en donde se crearán todas las entidades necesarias. Todas las entidades deben tener un *tipo de entidad o entity type* y en el presente caso adoptará el valor *UPCT:PARKING*.

En cuanto las propias entidades, se crearán 3 en total siguiendo lo expuesto en el Anexo I:

- UPCT:PARKING:ED (ED: Entrance/Exit Device) será la entidad encargada de controlar el tránsito general de vehículos. Cuenta con dos atributos: *car_in*, vinculada con el Dispositivo 1, y *car_out*, vinculada con el Dispositivo 2. Ambos son de tipo *value* e indican los coches que han entrado, los que han salido y, mediante su diferencia, las plazas ocupadas.

- UPCT:PARKING:D1 (D1: Device 1) corresponde a la entidad sincronizada con el Dispositivo 3, basado Arduino. Sus atributos serán *temperature, humidity* y *occupied* y sus tipos *centigrade, relative* y *value,* respectivamente. Estos indicarán las condiciones de temperatura y humedad de la plaza y si ésta está ocupada.

- UPCT:PARKING:D2 (D2: Device 2) es una entidad muy similar a la anterior, salvo que en este caso es el dispositivo basado en Raspberry Pi el que interactúa con ella. Los atributos de esta entidad son: *temperature, humidity*, *occupied* y *reserved.* Como se aprecia, se concederá al Dispositivo 4 la característica de poder ser reservado. El tipo de atributo *reserved* será también *value* al igual que ocurría con *occupied.*

Para la visualización del estado del parking y de todos los atributos mencionados se realizará una interfaz web mediante un widget que será incluido en la sección Mashup de FI-Lab. El nombre de dicho Widget será *UPCT Parking Interface.*

Finalmente, como aplicación de BigData y como uso de las numerosas posibilidades que hay, se va crear un histórico de datos en Cosmos de la entidad *UPCT:PARKING:ED,* de forma que sea posible consultar y hacer análisis del flujo de vehículos a lo largo de un periodo de tiempo definido por el usuario.

4.3 Dispositivo 1. Entrada al parking.

El dispositivo 1 es aquel que se encuentra en la entrada al sistema y tiene la función de contabilizar los vehículos que entran y señalizar cuándo hay plaza libre o no mediante el encendido de un LED verde o rojo.

4.3.1 Hardware.

Para llevar a cabo su función el dispositivo Arduino Uno en el que está basado este dispositivo llevará conectados los siguientes elementos:

- Wifi Shield de Arduino para permitir al dispositivo conectarse a Internet y comunicarse con FI-WARE.
- Sensor PIR HC-SR501 para la detección de movimiento a la entrada.
- Sensor SRF02 de ultrasonidos para corroborar que un coche está entrando mediante la medida de distancia.
- LEDs rojo y verde para indicar la ocupación del parking.

La conexión de los dispositivos se debe realizar teniendo en cuenta la disponibilidad de pines compatible con el Wifi Shield. En la en la Tabla 4.1 y en Ilustración 4.2 se muestra la conexión de los diferentes elementos.

Elemento	**Pin elemento**	**Pin Arduino**
SRF02	1	5V
	2	Analog 4
	3	Analog 5
	5	GND
HC-SR501	Power	5V
	Output	Digital 2
	GND	GND
Green Led	VCC	Digital 9
	GND	GND
Red Led	VCC	Digital 8
	GND	GND

Tabla 4.1. Conexiones de los elementos del dispositivo 1.

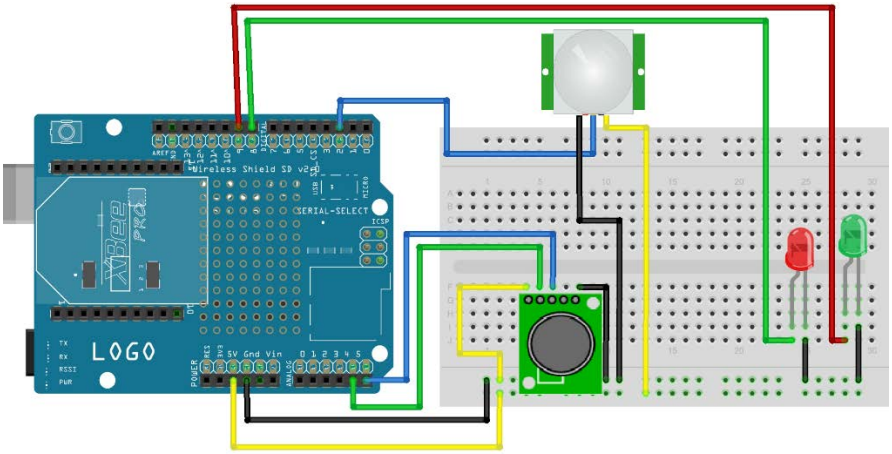

Ilustración 4.2. Vista de las conexiones del Dispositivo 1 creada con Fritzing

4.3.2 Vista del dispositivo real.

En la Ilustración 4.3 se muestra la implementación real del dispositivo 1.

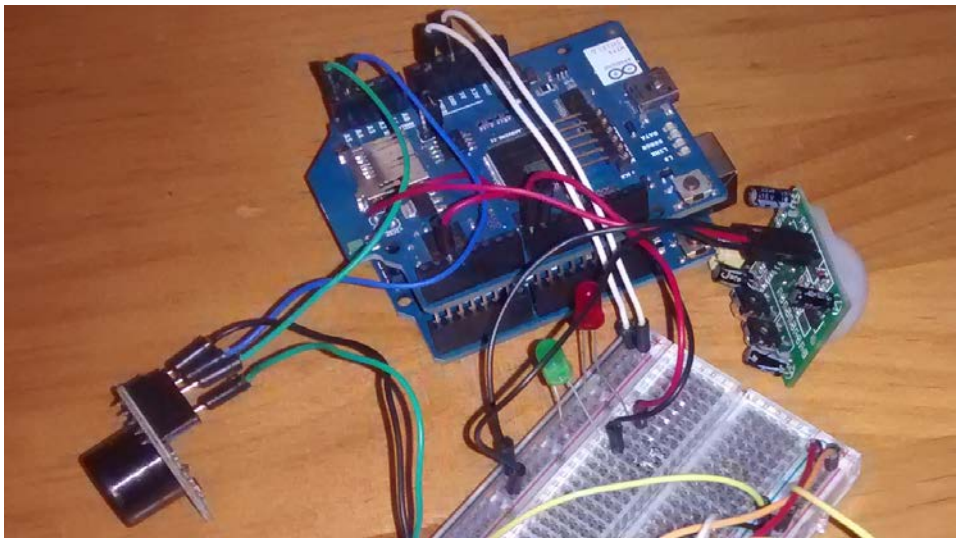

Ilustración 4.3. Vista real del dispositivo 1.

4.3.3 Software.

A continuación se detallan las partes características más importantes del Sketch de Arduino desarrollado. La Ilustración 4.4 muestra un esquema del funcionamiento del software. En el Anexo II se encuentran desarrolladas las funciones empleadas aquí.

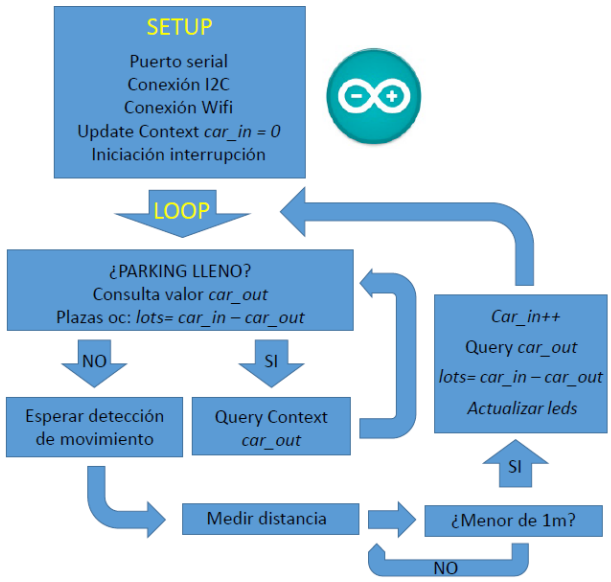

Ilustración 4.4. Esquema del software desarrollado para el dispositivo 1.

En primer lugar, se inicializan las variables necesarias. Para la conexión con el servidor es necesario inicializar la variable *IPAddress server()* con la IP pública de la instancia de FI-WARE.

```
char ssid[] = "SSID";
char pass[] = "Password";
int status = WL_IDLE_STATUS;
WiFiClient client;
IPAddress server(130,206,83,2);
String line="";
int red_led=8;
int green_led=9;
int reading = 0;
boolean detected;
int car_in=0;
int car_out=0;
int lots=0;
int max_car=14;
```

Se inicializa la transmisión serial para poder recibir información durante el proceso mediante *Serial.begin()*. Lo mismo para la conexión I^2C con *Wire.begin()*, ya que es la que utilizará el sensor SRF02. Además, se realiza la conexión con la red WiFi deseada habiendo declarado las variables *SSID* y *pass*. Tras los intentos necesarios, distanciados 10 segundos, se recibirá el mensaje *Wifi Ok* cuando se haya realizado correctamente la conexión.

```
Wire.begin();
Serial.begin(9600);

while (status != WL_CONNECTED) {
  status = WiFi.begin(ssid, pass);
  delay(10000);
}
Serial.println("Wifi OK");
```

El sensor PIR actuará con una interrupción, de forma que cuando éste detecte movimiento, pondrá en marcha el resto del código. Se declara la variable booleana *detected*

para señalar cuándo se ha realizado una detección y se inicializa la interrupción con la función *attachInterrupt()*. Seguidamente se actualiza el valor del atributo *car_in* de la entidad *UPCT:PARKING:ED,* que se inicializa a cero cada vez que Arduino arranca de nuevo mediante la función *updateContext(car_in),* donde *car_in* es una variable inicializada anteriormente. También se actualiza el estado de los LEDs mediante *updateLed(lots),* dependiente del número de plazas ocupadas y del número máximo de plazas del parking, *max_car*. Para ver el código de ambas funciones se puede consultar el Anexo II.

```
detected =false;
attachInterrupt(0,trig, RISING);
delay(1000);
updateContext(car_in);
updateLed(lots);
```

A partir de este punto comienza el *loop()* del sketch. Como ya se ha señalado, el código funciona con una interrupción disparada por el PIR. Sin embargo, ésta solo estará activa si hay plazas libres en el parking, por lo que se comprueba si se han llenado todas las plazas y si lo están se espera hasta que se libere espacio en intervalos de 15 segundos. La variable que define plazas ocupadas es *lot,* que es resultado de restar los coches que han entrado y han salido del parking. Para saber los coches que han salido se debe utilizar la función *queryContext(),* la cual pedirá el valor de contexto del atributo *car_out* de la entidad *UPCT:PARKING:ED*.

```
while(lots>=max_car){
    car_out=queryContext();
    lots=car_in-car_out;
    delay(15000);
  }
```

Si no están completas las plazas, el sistema queda pendiente de alguna detección por parte del sensor PIR. Concretamente, una vez se da dicha detección se comienza a medir la distancia mediante el sensor de ultrasonidos, de forma que cuando ésta se reduzca hasta un valor dado (por defecto 100 cm), indicando el paso de un coche, se mandará un mensaje avisando de que hay un coche nuevo en el parking. Se actualizará la variable *car_in,* ya que hay un coche que ha entrado y se recalcula en número de plazas ocupadas. Finalmente, se actualiza la entidad mediante la función *updateContext(),* así como el estado de los LEDs.

```
while(detected){
    reading=getDistance();
    Serial.println(reading);
    if(reading < 100){
      detected=false;
      attachInterrupt(0,trig,RISING); // 0-->pin2 1--Pin3
      Serial.println("There's a new car");
      car_in++;
      car_out=queryContext();
      lots=car_in-car_out;
      delay(2000);
      updateContext(car_in);
      updateLed(lots);
    }
Delay(250);
}
```

4.4 Dispositivo 2. Salida del parking.

El dispositivo 2 es aquel que se encuentra en la salida del sistema y tiene la función de contabilizar los vehículos que abandonan el parking, actualizando el valor del atributo *car_out* de la entidad *UPCT:PARKING:ED*.

4.4.1 Hardware.

El dispositivo 2 requiere menor número de elementos que el dispositivo 1, dado que no posee ningún tipo de señalización. Por lo tanto, los sensores que irán implementados en este dispositivo Arduino serán:

- Wifi Shield de Arduino para la permitir al dispositivo conectarse a Internet y comunicarse con FI-WARE.
- Sensor PIR HC-SR501 para la detección de movimiento a la salida.
- Sensor SRF02 de ultrasonidos para corroborar que un coche está saliendo mediante la medida de distancia.

En la en la Tabla 4.2 y en Ilustración 4.5 se muestra la conexión de los diferentes elementos.

Elemento	Pin elemento	Pin Arduino
SRF02	1	5V
	2	Analog 4
	3	Analog 5
	5	GND
HC-SR501	Power	5V
	Output	Digital 2
	GND	GND

Tabla 4.2. Conexiones de los elementos del dispositivo 2.

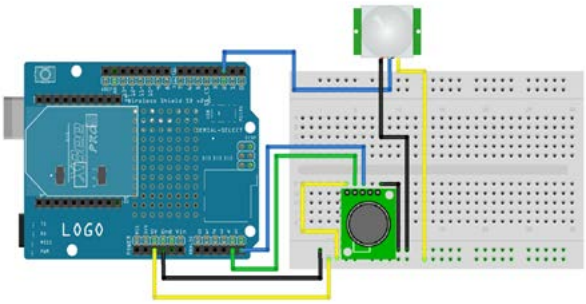

Ilustración 4.5. Vista de las conexiones del Dispositivo 2 creada con Fritzing.

4.4.2 Vista del dispositivo real.

En la Ilustración 4.6 se muestra el dispositivo 2 implementado. Como se puede observar, es muy similar al dispositivo 1 pero sin indicadores LED.

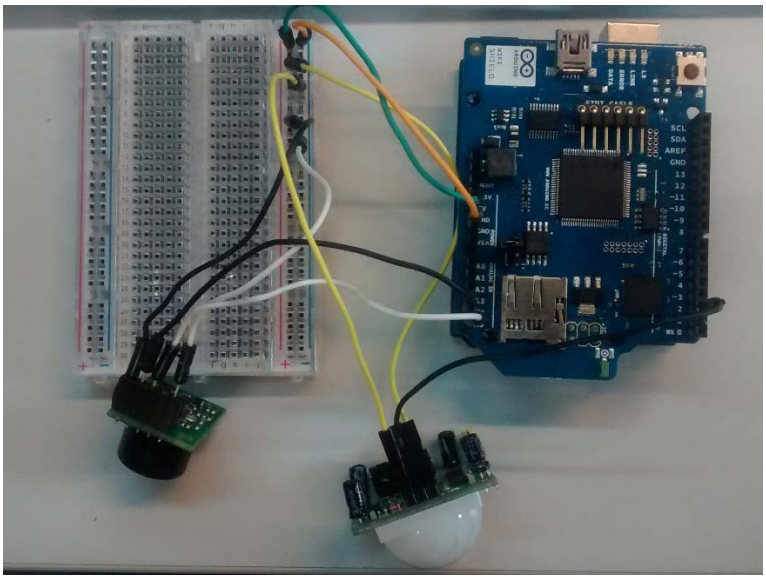

Ilustración 4.6. Vista real del dispositivo 2.

4.4.3 Software.

En la ilustración 4.7 aparece el esquema que muestra en líneas generales el funcionamiento del software.

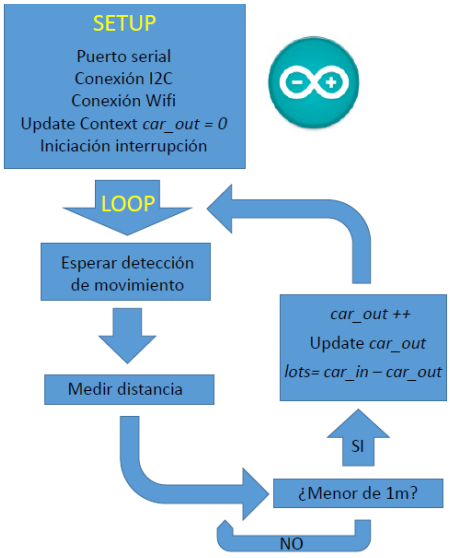

Ilustración 4.7. Esquema del software desarrollado para el dispositivo 2.

En primer lugar, se inicializan las variables necesarias.

```
char ssid[] = "SSID";
char pass[] = "Password";

int status = WL_IDLE_STATUS;

WiFiClient client;
IPAddress server(130,206,83,2);

int reading = 0;
boolean detected;
int car_out=0;
```

Al igual que en el dispositivo 1 se inician la transmisión serial e I^2C con *Serial.begin()* y *Wire.begin()*. Se realiza la conexión con la red WiFi deseada mediante intentos distanciados 10 segundos. El mensaje *Wifi Ok* indicará que dicha conexión se ha realizado satisfactoriamente.

```
  Wire.begin();
  Serial.begin(9600);

  while (status != WL_CONNECTED) {
    status = WiFi.begin(ssid, pass);
    delay(10000);
  }
  Serial.println("Wifi OK");
```

El sensor PIR actuará con una interrupción, de forma que cuando éste detecte movimiento pondrá en marcha el resto del código. Se declara la variable booleana *detected* para señalar cuándo se ha realizado una detección y se inicializa la interrupción con la función *attachInterrupt()*. Finalmente, se actualiza el valor de contexto del atributo *car_out*. Como ocurría en la entrada del parking, cada vez que el dispositivo es reiniciado se inicializa a cero.

```
  detected =false;
  attachInterrupt(0,trig, RISING);// 0-->pin2 1--Pin3
  delay(1000);
  updateContext(car_out);
```

Entonces el dispositivo queda pendiente de una interrupción por parte del sensor PIR. Una vez se da dicha detección se comienza a medir la distancia mediante el sensor de ultrasonidos, de forma que cuando ésta se reduzca hasta un valor de 100 cm, indicando el paso de un coche, se enviará un mensaje avisando de que hay un coche saliendo del parking. Se actualizará la variable *car_out* y se enviará a la entidad de FI-WARE mediante la función *updateContext()*, comentada anteriormente.

```
while(detected){
    reading=getDistance();
    Serial.println(reading);
    if(reading < 100){
      detected=false;
      attachInterrupt(0,trig,RISING);
      Serial.println("There's a car leaving the parking");
      car_out+=1;
      updateContext(car_out);
    }
    delay(250);
  }
```

4.5 Dispositivo 3. Plaza de aparcamiento.

El dispositivo 3 es uno de los dos modelos de dispositivo desarrollados para ser ubicados en las plazas del parking. Tiene la función de indicar si la plaza está libre o no y proporcionar datos climatológicos de temperatura y humedad relativa.

4.5.1 Hardware.

Los elementos implementados en el dispositivo se enumeran a continuación.

- WiFi Shield de Arduino para la permitir al dispositivo conectarse a Internet y comunicarse con FI-WARE.
- Sensor SHT11 para medición de temperatura y humedad relativa.
- Sensor SRF02 de ultrasonidos para comprobar si existe un coche o no en la plaza.
- LEDs rojo y verde para indicar la ocupación de la plaza.

En la en la Tabla 4.3 y en Ilustración 4.8 se muestra la conexión de los diferentes elementos al Arduino Uno.

Elemento	Pin elemento	Pin Arduino
SRF02	1	5V
	2	Analog 4
	3	Analog 5
	5	GND
SHT11	VCC	5V
	CLK	Digital 5
	DATA	Digital 6
	GND	GND
Green Led	VCC	Digital 9
	GND	GND
Red Led	VCC	Digital 8
	GND	GND

Tabla 4.3. Conexiones de los elementos del dispositivo 3.

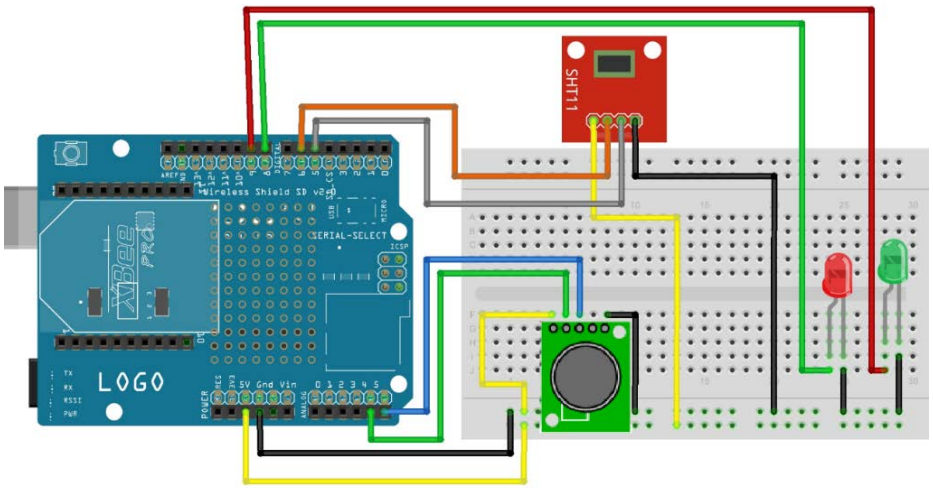

Ilustración 4.8. Vista de las conexiones del Dispositivo 3 creada con Fritzing.

4.5.2 Vista del dispositivo real.

En la Ilustración 4.9 se muestra la vista del dispositivo 3, donde se observa la conexión del hardware.

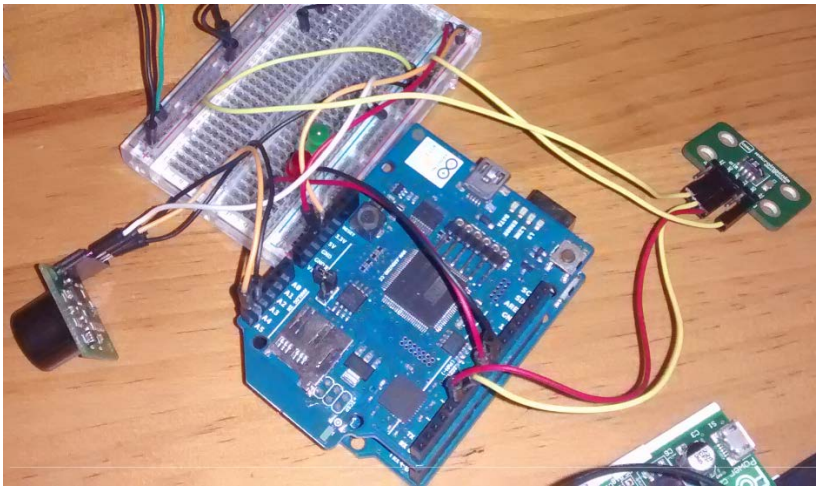

Ilustración 4.9. Vista real del dispositivo 3.

4.5.3 Software.

La Ilustración 4.10 muestra el esquema general del software desarrollado para el dispositivo 3.

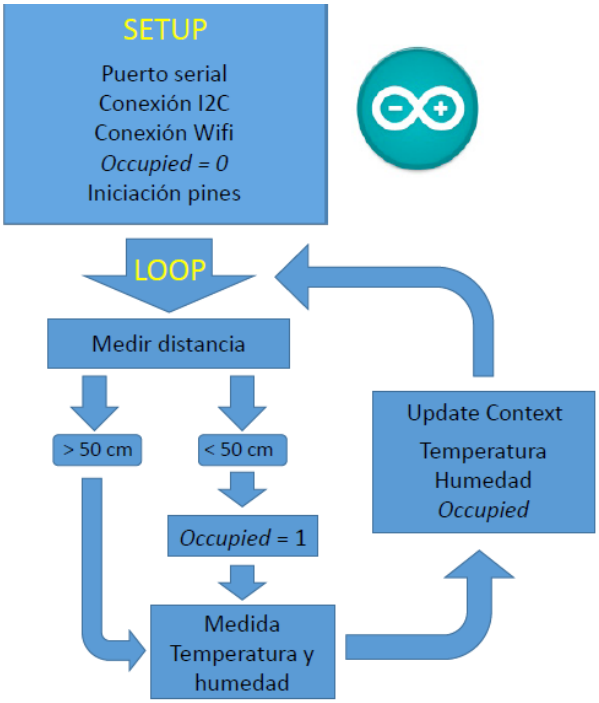

Ilustración 4.10. Esquema del software desarrollado para el dispositivo 3.

Para el uso del sensor SHT11, o cualquiera de la misma serie de la marca *Sensirion,* se importa una librería para Arduino de la misma marca distribuida en la Web de Arduino. Dicha librería es importada mediante la sentencia *#include <sensirion.h>*.

En este caso las variables inicializadas son las mostradas a continuación.

```
char ssid[] = "SSID";
char pass[] = "Password";

int status = WL_IDLE_STATUS;

WiFiClient client;
IPAddress server(130,206,83,2);

int red_led=8;
int green_led=9;

const uint8_t dataPin  =  6;
const uint8_t clockPin =  5;
float temperature;
float humidity;
float dewpoint;
Sensirion tempSensor = Sensirion(dataPin, clockPin);

int reading[] = {0,0,0};
float average;
int occupied = 0;
int update_time=10;
int count=0;
```

En este caso, el dispositivo realizará continuamente la misma operación, que es medir la distancia desde el sensor de ultrasonidos para comprobar cuando hay o no un vehículo aparcado. Esta comprobación se realizará cada 3 segundos.

Para obtener una mayor fiabilidad de la medida de distancia, se realizan en cada ejecución del bucle *loop()* tres medidas seguidas que se almacenan en el array *read[ ]* y posteriormente se haya su media y se guarda en la variable real *average*.

```
for(int i=0;i<3;i++){
    reading[i]=getDistance();
    Serial.println(reading[i]);
    delay(150);
  }
average =(reading[0]+reading[1]+reading[2])/3;
```

Tras la obtención de la distancia, se comprueba si esta medida es mayor o menor de 50 cm (por defecto), lo que implicará que haya o no un vehículo estacionado, y por tanto, que el valor de la variable *occupied* cambie. Seguidamente se actualiza el estado de los LEDs.

```
if(average<50){
   occupied=1;
  }else{
    occupied=0;
  }
updateLed(occupied);
```

Después de llevar a cabo esta comprobación, se obtienen la temperatura y humedad relativa actuales y se envía al monitor serial. Con estos valores, y junto con la variable de ocupación obtenida, se actualizan los valores de los respectivos atributos. Mediante la modificación de la variable *update_time* y gracias al contador incluido con la variable *count* se puede modificar el tiempo entre actualizaciones enviadas a FI-WARE. El valor de *update_time* será el número de comprobaciones que el dispositivo realizará antes de ejecutar la función *updateContext()*. Esta explicación se puede complementar con la incluída en el Anexo II.

```
  tempSensor.measure(&temperature, &humidity, &dewpoint);
  Serial.print("Temperature: ");
  Serial.println(temperature);
  Serial.print("Humidity: ");
  Serial.println(humidity);
  Serial.print("occupied: ");
  Serial.println(occupied);
  if(count>=update_time){
    updateContext(temperature,humidity,occupied);
    count=0;
  }
  count++;
  delay(3000);
```

4.6 Dispositivo 4. Plaza de aparcamiento.

El dispositivo 4 es el segundo de los dos modelos de dispositivo desarrollados para ser ubicados en las plazas del parking. A diferencia de los otros tres expuestos, éste está basado en un dispositivo empotrado Raspberry Pi. Tiene la función de indicar si la plaza está libre o no, proporcionar datos climatológicos de temperatura y humedad relativa y posibilitar la reserva del aparcamiento.

4.6.1 Hardware.

En este dispositivo la conexión a Internet de realiza mediante cable Ethernet por lo que no es necesario ninguna configuración WiFi, tanto de software como de hardware. El resto de componentes implementados son los siguientes.

- Sensor SHT11 para medición de temperatura y humedad relativa.
- Sensor HC-SRF04 para comprobar si existe un coche o no en la plaza.
- LEDs rojo y verde para indicar la ocupación de la plaza.

Es importante mencionar que la conexión del sensor de temperatura y humedad relativa SHT11 necesita una conexión extra de una resistencia de *pull-up* de 10 kΩ.

En la en la Tabla 4.4 y en Ilustración 4.11 se describe la conexión necesaria.

Elemento	Pin elemento	Pin RPi	GPIO
HC-SR04	VCC	Pin 2	
	Trig	Pin 11	GPIO 17
	Echo	Pin 16	GPIO 23
	GND	GND	
SHT11	VCC	Pin 1	
	CLK	Pin 15	GPIO 22
	DATA	Pin 22	GPIO 25
	GND	GND	
	Resistencia 10 kΩ DATA-VCC		
Green Led	VCC	Pin 24	GPIO 8
	GND	GND	
Red Led	VCC	Pin 18	GPIO 24
	GND	GND	

Tabla 4.4. Conexiones de los elementos del dispositivo 4.

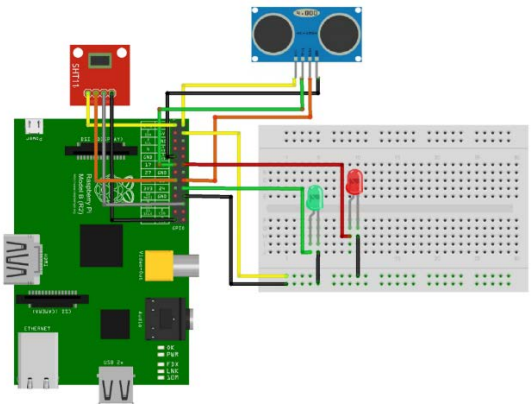

Ilustración 4.11. Vista de las conexiones del Dispositivo 4 creada con Fritzing.

4.6.2 Vista real del dispositivo.

En la Ilustración 4.12. se muestra la vista del dispositivo 4. En este caso, basado en Raspberry Pi.

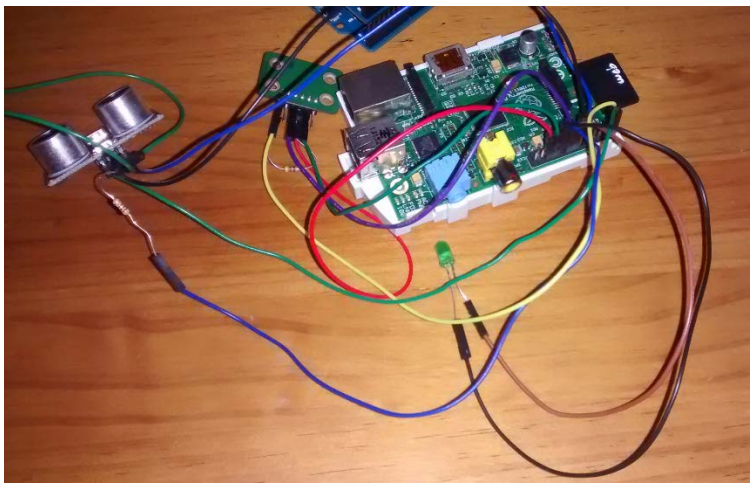

Ilustración 4.12. Vista real del dispositivo 4.

4.6.3 Software.

En este caso, la programación del dispositivo empotrado se realiza con lenguaje de programación *C++*. El esquema de funcionamiento se muestra en la Ilustración 4.13 [10].

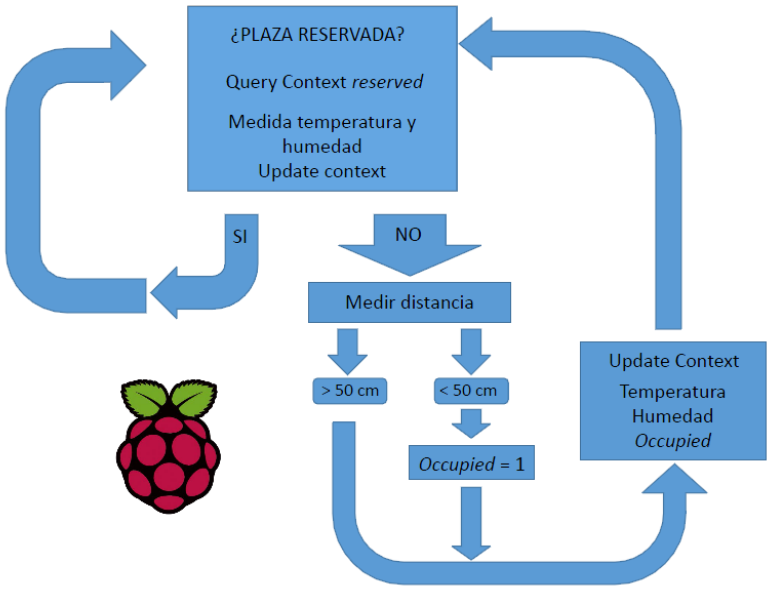

Ilustración 4.13. Esquema de funcionamiento del dispositivo 4.

La utilización del GPIO de Raspberry Pi se puede realizar desde la propia línea de comandos. Sin embargo, existe una librería creada bajo licencia de software libre para el procesador *bcm2835* llamada *lbcm2835*. Con ella el uso de los pines I/O de este mini-PC es mucho más sencillo. Toda la información sobre ella se encuentra en la web *http://www.airspayce.com/mikem/bcm2835/*.

La programación de Raspberry Pi se realizará en varios archivos para simplificar el código y hacerlo más entendible. Así, el software del dispositivo se divide en:

- *clientSocketHttp6.cpp* y *clientSocketHttp.h*. Son dos archivos que contienen todo lo necesario para establecer la comunicación por HTTP con FI-WARE. Se encuentran disponibles en el git hub *live demo* de FI-WARE.
- *RPi_SHT1x.cpp* y *RPi_SHT1x.h*. Contienen sentencias necesarias para hacer funcionar el sensor SHT11 en el dispositivo. Su contenido se encuentra detallado en el Anexo II.
- *disp4.cpp*. Es el archivo que contiene el código principal del sistema análogo a los Sketches vistos en los demás dispositivos del estudio.

Por lo tanto, los detalles del código del archivo *disp4.cpp* son los de interés en este capítulo, dado que explican el funcionamiento general del dispositivo.

En primer lugar, se definen los pines GPIO que se van a utilizar y se indican las funciones que se van a utilizar a lo largo del código, detalladas en el Anexo II.

```
#define BCM2835_CORE_CLK_HZ 250000000
#define PIN_TRIG RPI_GPIO_P1_11
#define PIN_ECHO RPI_GPIO_P1_16
#define GREEN_LED RPI_GPIO_P1_24
#define RED_LED RPI_GPIO_P1_18
#define MAX_SML_BUFF 10000

std::string updateContext(float,float,int,int);
std::string queryContext();
void updateLed(int);
float getTemp(void);
float getHumid(void);
```

Seguidamente se inicia la función principal *main()* y se declaran las variables globales.

```
        std::stringstream strings;
        std::string sendData;
        std::string replyData;
        std::string resourceDirQuery;
        std::string resourceDirUpdate;
        bcm2835_init();
        int occupied;
        int reserved;
        float reading[3]={};
        float average;
        float temp,humid;
        int m1,m2;
        resourceDirQuery = "/ngsi10/queryContext";
        resourceDirUpdate="/ngsi10/updateContext";
```

Para conseguir un bucle que haga que el dispositivo funcione indefinidamente se utiliza la sentencia *while(1)*.

A continuación, la primera comprobación que se realiza es si la plaza está reservada. Para ello, en cada iteración, se ejecuta una sentencia *do while,* que hace esperar al sistema hasta que la variable *reserved* cambia su valor a 0. Para ese fin, se va actualizando su valor con el atributo de la entidad *UPCT:PARKING:D2*. Para realizar dicha actualización, el sistema ejecuta dos operaciones. En primer lugar, carga el formulario XML correspondiente al *queryContext* en la variable de tipo string *sendData* y se envía mediante la función *sendHttpSockett()*. En segundo lugar, se utilizan una serie de sentencias para filtrar de la respuesta del servidor, el valor de contexto que se guardará en la variable *reserved*.

```
sendData=queryContext();

replyData =sendHttpSocket("130.206.83.2",1026,"POST",resourceDirQuery,std::\
string("application/xml"),sendData);

int pos1 = replyData.find("contextValue");
pos1 =pos1 + 13;
int pos2 =replyData.find("</contextValue");
int size = pos2-pos1;
strings<<replyData.substr(pos1,size);
strings>>reserved;
```

Tras conocer si está reservada o no la plaza, se actualiza el estado de los LEDs y se comienza con la obtención de temperatura y humedad relativa. Estos valores se envían siguiendo el procedimiento anterior utilizando en este caso la función *updateContext()* para la variable *sendData*. También se actualizarán los valores de contexto de las variables *occupied* y *reserved,* dado que en las siguientes iteraciones éstos habrán sido actualizados. Finalmente, una función *delay()* hace que transcurran 10 segundos entre iteraciones.

```
updateLed(reserved);
printf("Updating FI-WARE info...\n");
temp=getTemp();
humid=getHumid();
sendData=updateContext(temp,humid,occupied,reserved);
replyData =sendHttpSocket("130.206.83.2",1026,"POST",resourceDirUpdate,std:
:string("application/xml"),sendData);
```

Aquí termina la sentencia *do* del bucle *do while*. Si está reservado el estacionamiento se volverá a iniciar el bucle, en caso contrario el programa continuará.

Al igual que en el dispositivo 3, se realizan tres mediciones que se alojarán en el array *reading* y posteriormente se calculará su media para obtener una mayor fiabilidad. La sentencia que obtiene las tres medidas se muestra en el Anexo II.

Si la media, guardada en la variable *average* es mayor de 50 cm (por defecto), la variable *occupied* será 0 (será 1 cuando sea menor). Posteriormente, se actualiza el estado de los LEDs

y se ejecuta la función *delay(2000),* para hacer que transcurran 2 segundos hasta que se inicie de nuevo el bucle *while(1).*

```
            average=(reading[0]+reading[1]+reading[2])/3;
            if(average<0.5){
                printf("occupied lot\n");
                occupied=1;
            }else{
                printf("Free lot\n");
                occupied=0;
            }
            updateLed(occupied);
            delay(2000);
            }
        bcm2835_close();
```

La compilación del software en Raspberry Pi debe realizarse bajo permisos de usuario root mediante el código siguiente, creándose el ejecutable *disp4*.

```
        g++ disp4.cpp clientSocketHttp.cpp RPi_SHT1x.cpp –lbcm2835 –o disp4
```

4.7 Persistencia de datos. Cosmos.

Como aplicación de persistencia de datos, que *Cosmos* permite conseguir para almacenar históricos de datos, se va a realizar un histórico de datos de los vehículos que entran y abandonan el parking.

Para ello, la máquina virtual *UPCT,* debe estar configurada como se muestra en el Anexo I. Posteriormente, debe realizarse una suscripción de la entidad *UPCT:PARKING:ED,* alojada en la máquina virtual, con el inyector *Cygnus* instalado anteriormente en la misma máquina virtual. Esto hará que los datos queden alojados cada vez que haya una variación en el valor de contexto junto con la fecha y hora exactas en la que sucedió.

Para realizar la suscripción, se debe enviar a la instancia el formulario XML correspondiente mostrado a continuación.

```
        <?xml version="1.0"?>
        <subscribeContextRequest>
          <entityIdList>
            <entityId type="UPCT:PARKING" isPattern="false">
              <id>UPCT:PARKING:ED</id>
            </entityId>
          </entityIdList>
          <reference>http://localhost:5050/notify</reference>
          <duration>P1M</duration>
          <notifyConditions>
            <notifyCondition>
              <type>ONCHANGE</type>
              <condValueList>
                <condValue>car_in</condValue>
              </condValueList>
            </notifyCondition>
          </notifyConditions>
          <throttling>PT5S</throttling>
        </subscribeContextRequest>
```

Dichos datos se alojarán en formato JSON en un archivo de texto HDFS en la dirección de cosmos *hdfs:///user/quiquehz/org42/UPCT_PARKING_ED-UPCT_PARKING/UPCT_PARKING_ED-UPCT_PARKING.txt*.

Es posible visualizar los datos guardados accediendo a la máquina virtual de Cosmos (ver Anexo I apartado I.5.2) y ejecutando el comando siguiente:

```
hadoop fs -cat /user/quiquehz/org42/UPCT_PARKING_ED-UPCT_PARKING/UPCT_PARKING_ED-
UPCT_PARKING.txt
```

La respuesta obtenida se muestra a continuación

```
{"recvTime":"2014-09-15T17:09:14.536","car_out":"3", "car_out_md":[],"car_in":"6", "car_in_md":[]}
{"recvTime":"2014-09-15T17:09:50.501","car_out":"3", "car_out_md":[],"car_in":"3", "car_in_md":[]}
{"recvTime":"2014-09-15T17:12:50.286","car_out":"3", "car_out_md":[],"car_in":"4", "car_in_md":[]}
{"recvTime":"2014-10-03T12:30:48.531","car_out":"3", "car_out_md":[],"car_in":"0", "car_in_md":[]}
{"recvTime":"2014-10-03T12:34:35.2","car_out":"3", "car_out_md":[],"car_in":"1", "car_in_md":[]}
{"recvTime":"2014-10-03T12:34:45.975","car_out":"3", "car_out_md":[],"car_in":"2", "car_in_md":[]}
{"recvTime":"2014-10-03T12:35:17.926","car_out":"3", "car_out_md":[],"car_in":"0", "car_in_md":[]}
{"recvTime":"2014-10-03T12:35:36.921","car_out":"3", "car_out_md":[],"car_in":"1", "car_in_md":[]}
{"recvTime":"2014-10-04T11:54:14.97","car_out":"2", "car_out_md":[],"car_in":"0", "car_in_md":[]}
{"recvTime":"2014-10-04T11:54:23.588","car_out":"2", "car_out_md":[],"car_in":"1", "car_in_md":[]}
{"recvTime":"2014-10-05T12:14:30.591","car_out":"0", "car_out_md":[],"car_in":"12", "car_in_md":[]}
{"recvTime":"2014-10-05T14:05:43.064","car_out":"0", "car_out_md":[],"car_in":"13", "car_in_md":[]}
{"recvTime":"2014-10-05T14:06:01.536","car_out":"0", "car_out_md":[],"car_in":"14", "car_in_md":[]}
{"recvTime":"2014-10-05T14:07:27.126","car_out":"0", "car_out_md":[],"car_in":"12", "car_in_md":[]}
{"recvTime":"2014-10-05T14:17:07.101","car_out":"0", "car_out_md":[],"car_in":"13", "car_in_md":[]}
{"recvTime":"2014-10-05T14:18:23.524","car_out":"0", "car_out_md":[],"car_in":"12", "car_in_md":[]}
{"recvTime":"2014-10-05T14:19:59.016","car_out":"0", "car_out_md":[],"car_in":"13", "car_in_md":[]}
{"recvTime":"2014-10-05T14:21:13.807","car_out":"0", "car_out_md":[],"car_in":"14", "car_in_md":[]}
{"recvTime":"2014-10-05T14:22:04.013","car_out":"0", "car_out_md":[],"car_in":"12", "car_in_md":[]}
{"recvTime":"2014-10-05T14:22:39.219","car_out":"0", "car_out_md":[],"car_in":"13", "car_in_md":[]}
{"recvTime":"2014-10-05T14:24:11.9","car_out":"0", "car_out_md":[],"car_in":"14", "car_in_md":[]}
{"recvTime":"2014-10-05T14:25:12.984","car_out":"0", "car_out_md":[],"car_in":"13", "car_in_md":[]}
{"recvTime":"2014-10-05T14:26:29.246","car_out":"0", "car_out_md":[],"car_in":"12", "car_in_md":[]}
{"recvTime":"2014-10-05T14:27:18.21","car_out":"0", "car_out_md":[],"car_in":"13", "car_in_md":[]}
{"recvTime":"2014-10-05T14:29:45.798","car_out":"0", "car_out_md":[],"car_in":"14", "car_in_md":[]}
{"recvTime":"2014-10-05T14:30:19.493","car_out":"0", "car_out_md":[],"car_in":"13", "car_in_md":[]}
```

Como se puede apreciar, en Cosmos se guardan los valores de los atributos de la entidad suscrita al inyector Cygnus, la fecha y hora del cambio de cada valor y los valores de metadata de cada atributo, nulos en este caso.

4.8 Interfaz web. Mashable application widget.

La aplicación creada es ejecutable en la sección Mashup de FI-Lab y su función es la visualización de los datos aportados por el parking inteligente.

En esta sección se muestra la interfaz gráfica del widget y se explica su funcionamiento. La estructura de la aplicación y el desarrollo de los archivos que la componen el widget se muestran en detalle en el Anexo II.

UPCT Parking Interface es el nombre de esta aplicación y su aspecto se muestra en la Ilustración 4.14.

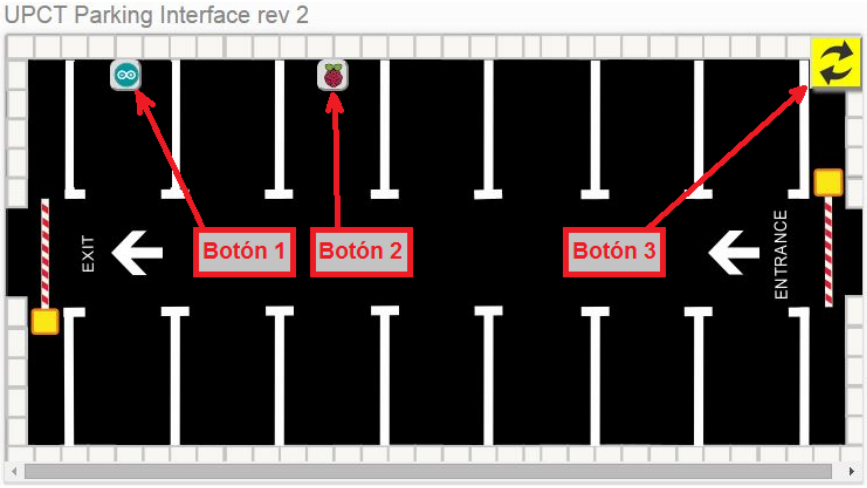

Ilustración 4.14. Vista de la interfaz del widget y sus botones principales

En el widget se puede observar el esquema análogo al parking real del caso de estudio. Como se mencionó en la descripción general, el parking cuenta con 14 plazas, pero por simplicidad se han desarrollado sólo 4 dispositivos, de los cuales son 2 los que se ubican en una plaza del parking. En la Ilustración 4.14 se observa cuáles son esos dos dispositivos, identificados por un botón identificativo (botones 1 y 2). En el caso de implementar el sistema en un parking real, todas las plazas tendrían un botón propio.

Cada botón tendrá la función que se muestra a continuación.

- Botón 1. Al pulsarlo aparece en el centro del widget el último valor de temperatura y humedad relativa registrado por el dispositivo 3, además de su nombre (ver Ilustración 4.15)

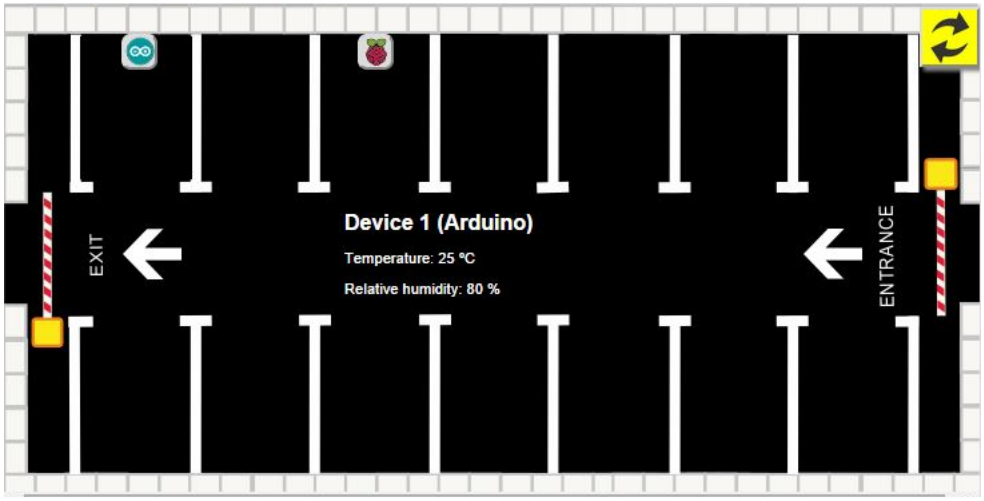

Ilustración 4.15. Vista del widget al pulsar el botón 1.

- Botón 2. Al pulsarlo aparece en el centro del widget el último valor de temperatura y humedad relativa registrado por el dispositivo 4, además de su nombre (ver Ilustración 4.16)

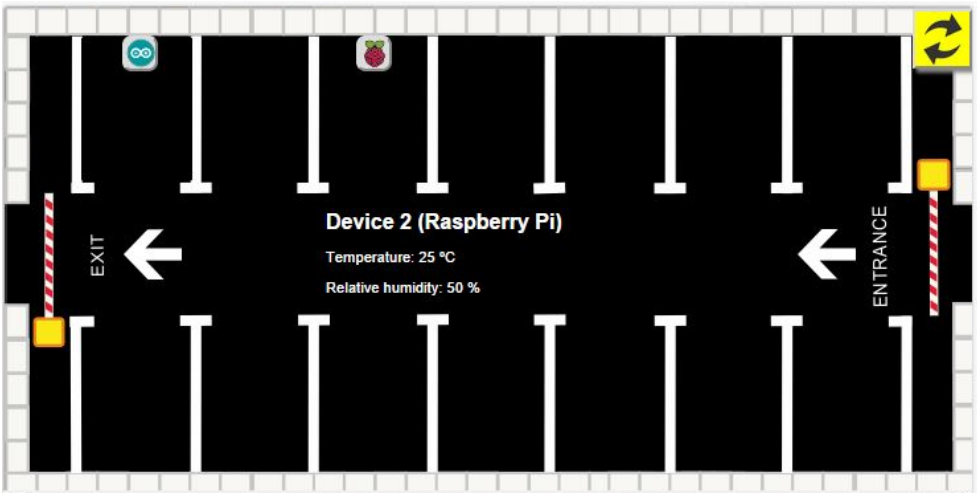

Ilustración 4.16. Vista del widget al pulsar el botón 2.

- Botón 3. Se trata de un botón de actualización del estado. Al pulsarlo aparece en el centro del widget el número de plazas disponibles en el parking, como resultado de la diferencia entre el número de plazas totales (14) y el balance *car_in* - *car_out* de la entidad (ver Ilustración 4.17). Además, en cada una de las plazas aparecerá una imagen distinta dependiendo del estado de la plaza. En la Ilustración 4.15 y 4.16 aparecen las dos plazas libres mientras que en la Ilustración 4.17 se muestra una plaza ocupada y la otra reservada.

Ilustración 4.17. Vista del widget al pulsar el botón 3.

Capítulo 5

Conclusiones y Trabajos futuros.

5.1 Conclusiones

En este proyecto se ha abordado el estudio de la plataforma FI-WARE. Concretamente, se ha diseñado un parking inteligente, basado en dispositivos empotrados de bajo coste, eficiente y que podría resultar de gran utilidad, tanto para el usuario como para el gerente de la empresa. Dicho parking ha sido el caso de estudio seleccionado para estudiar la plataforma.

Se ha realizado un estudio para escoger los dispositivos empotrados y la plataforma de Internet de las Cosas más adecuada. Para los dispositivos empotrados, se ha tenido en cuenta el coste, su facilidad de integración, sus prestaciones y su grado de extensión en el panorama actual. Se han seleccionado Raspberry Pi y Arduino. En el caso de la plataforma para Internet de las Cosas, se han tenido en cuenta el coste, la importancia de la plataforma y su potencial. FI-WARE ha sido elegida por ser la plataforma elegida por Europa para la creación de Ciudades Inteligentes y por no requerir de inversión monetaria para su uso.

Se ha llevado a cabo un caso de estudio en el que se ha diseñado y ejecutado un modelo de parking inteligente. Para ello, se ha implementado el hardware necesario mediante la integración de sensores a los dispositivos empotrados y se ha obtenido el software mediante el desarrollo de los archivos pertinentes. En el caso de Arduino, se han creado todos los Sketches necesarios, y para Raspberry Pi, se ha utilizado el lenguaje *C++* para

implementar todos los componentes software necesarios. Finalmente, se ha creado una aplicación Web que aporta una interfaz gráfica al sistema. Con ella se pueden llevar a cabo la visualización de parámetros del parking, tales como temperatura, humedad relativa, ocupación, plazas libres y ubicación de dichas plazas. Se han empleado para ello los lenguajes de etiquetas HTML, CSS y XML, y los lenguajes Javascript y jQuery. Dicha aplicación o widget ha sido generado para ser compatible con la plataforma *Wirecloud*.

Finalmente y a modo de resumen, con este estudio se ha pretendido comprobar la utilidad de uno de los muchos servicios posibles que pueden ser creados mediante esta plataforma. La creciente evolución de las ciudades hacia la inteligencia en la nube, da muestra del verdadero potencial de ésta tecnología. FI-WARE es una plataforma que, aun estando en desarrollo, cumple fielmente los objetivos para los que ha sido creada. Es además la más flexible de todas las plataformas estudiadas.

5.2 Trabajos Futuros

Como posible trabajo futuro, se puede crear una aplicación de mayor dimensión mediante la implementación de otros sensores que aporten información complementaria, como leer la matrícula de los vehículos. Por otro lado, se pueden llevar a cabo estudios más exhaustivos de optimización mediante el análisis del *BigData* almacenado en *Cosmos*. Es muy interesante también la posibilidad del pago online antes del acceso al parking, aprovechando la ya implementada función de reserva. Incluso, sería posible visualizar y filtrar en la interfaz web el vídeo captado por cámaras de seguridad mediante el GE *Kurento,* que FI-WARE ha desarrollado.

El caso de estudio llevado a cabo es completamente funcional y es posible implementarlo en el parking de cualquier ciudad. Sin embargo, como se ha citado anteriormente, esta es una de las muchas aplicaciones que FI-WARE permite crear. Son innumerables las posibilidades que ésta plataforma pone en manos de los desarrolladores para la creación de ciudades innovadoras que hagan más sencilla, segura y eficiente la vida en ellas. Éste es otro de los aspectos que FI-WARE fomenta, mediante la creación de *Hackatones* y concursos de ideas emprendedoras.

Anexo I

Documentación y herramientas para la utilización de Orion CB y Cosmos

I.1 Introducción.

En este anexo se recoge todo lo necesario para comenzar a trabajar con la plataforma FI-WARE, desde la creación de una máquina virtual en FI-LAB hasta la configuración de ésta para conseguir la persistencia de datos en históricos.

Las instalaciones y utilidades mostradas son las empleadas para la creación posterior del caso de estudio.

I.2 Creación de una instancia Orion.

El primer paso para la creación de una instancia en una máquina virtual en FI-WARE es el registro del usuario en el portal FI-LAB en la dirección https://account.lab.fi-ware.org/.

Una vez completado, se debe acceder a la sección Cloud del mismo portal, ya que es ahí donde se realizará el resto del proceso.

El primer requisito para crear una instancia *Context Broker* es generar una clave de acceso o *keypair* de seguridad que permitirá el acceso a múltiples máquinas. Dicha clave es de

extensión *.pem* y se generará con el nombre indicado. La creación de estas claves se realiza en el apartado *Security* de la columna izquierda y dentro de la pestaña *Keypairs* hacer click en *Create Keypair* (Ver Ilustración I.1).

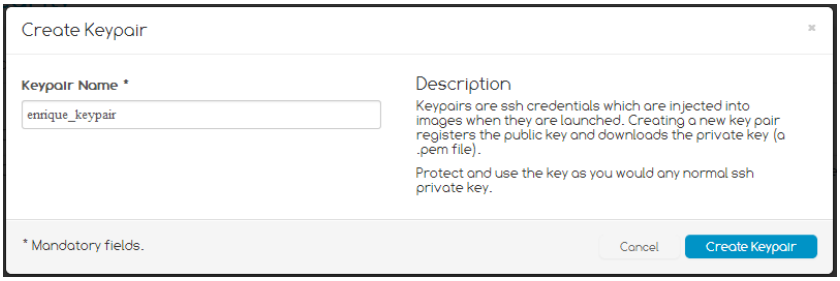

Ilustración I.1. Vista de la ventana emergente para generación de keypairs.

En este mismo apartado de seguridad se configuran los grupos de seguridad o *security groups*. Por defecto, aparece uno válido llamado *default*. En su configuración se pueden añadir reglas para seleccionar los puertos que la máquina virtual tendrá abiertos. Dado que las entidades por defecto se crean a través del puerto 1026, y que, el futuro acceso remoto a la máquina se realizará mediante conexión SSH, deben permanecer abiertos los puertos 1026 y 22 (ver Ilustración I.2).

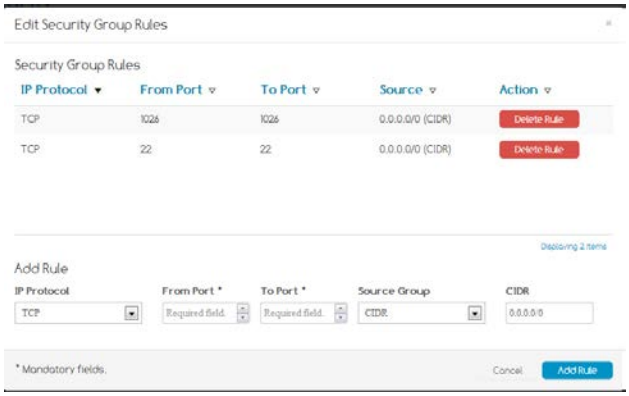

Ilustración I.2. Vista de la ventana de configuración de las reglas de los grupos de seguirdad.

Como último paso en la sección de seguridad, es necesario generar una IP pública a asociar posteriormente a la instancia para poder acceder a ella remotamente. Ello se lleva a cabo en el apartado *Floating IP*. La IP asignada al caso de estudio del presente proyecto es 130.206.83.2 (ver Ilustración I.3).

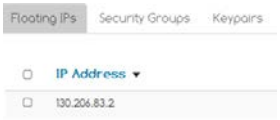

Ilustración I.3. Vista de la IP pública generada.

Como se expone en el Capítulo 3, existen dos formas de instanciar una máquina virtual. Aquí se detalla la instalación de la imagen *orion-psb-image*, basada en el SO Linux *CentOS*.

Las imágenes se encuentra en la sección *images* del menú *compute*. Para iniciar la instalación se hace click sobre el botón *launch* (ver Ilustración I.4).

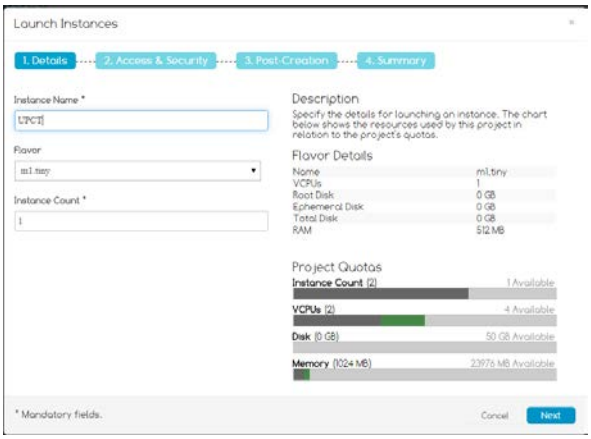

Ilustración I.4. Vista del primer paso de la ejecución de la orion-psb-image.

Durante el proceso de instalación se deben seleccionar tanto el *keypair* generado como el grupo de seguridad *default* (ver Ilustración I.5).

Ilustración I.5. Vista del segundo paso de la ejecución de la orion-psb-image.

Los dos siguientes pasos no son necesarios para el objetivo de este proyecto, aunque es de interés conocer que tras la instalación de la imagen es posible configurar *scripts* para conseguir un funcionamiento específico.

Finalmente, se asocia la IP pública antes conseguida con la instancia *UPCT* creada (ver Ilustración I.6).

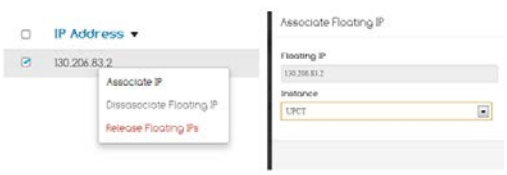

Ilustración I.6. Vista de la asociación de IP pública con la instancia.

En la Ilustración I.7 se muestra el resultado final de la instalación en el panel *Instances* del portal FI-LAB. En dicho panel se muestra información sobre la máquina y su estado.

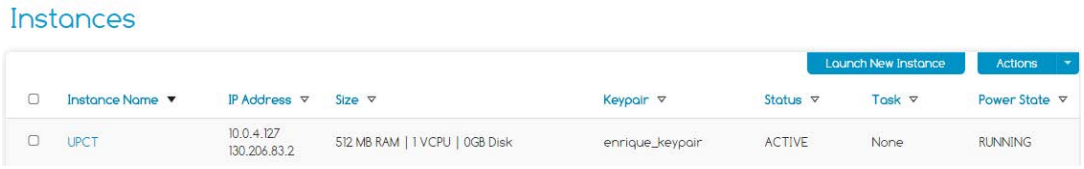

Ilustración I.7. Vista del panel Instances de FI-LAB

I.3 Utilización de Orion. Create, update, query and subscribe entities.

Gracias a la API creada por los desarrolladores de la plataforma, es posible interactuar con la máquina mediante peticiones *REST* como se ha documentado en el Capítulo 3. [11]

A continuación se exponen las diferentes peticiones que deben enviarse mediante *POST* y las respuestas del servidor ante ellas. A modo de ejemplo, se va a operar con una entidad llamada *UPCT:TEMPERATURE:SENSOR*, de tipo *UPCT:SENSOR* y con los atributos *temperature* y *humidity*.

La instancia donde se guarde dicha entidad será la creada anteriormente, y, por tanto, el envío de datos se debe realizar a la dirección 130.206.83.2:1026. Además, dependiendo de la operación a realizar debe realizarse a una ruta específica.

- Creación y actualización de entidades, */NGSI10/updateContext*.
- Consulta (*query*) de información, */NGSI10/queryContext*.
- Suscripción entre entidades, */NGSI10/subscribeContext*.

I.3.1 Creación de una entidad. Entity creation.

En este caso, se desea crear la entidad *UPCT:TEMPERATURE:SENSOR* con los atributos *temperature y humidiy* (unidades en grados centígrados y tanto por ciento de humedad relativa, respectivamente) y valores de contexto de 25 ºC y 60 %. Para ello, el formulario XML necesario que debe ser enviado a la dirección *130.206.83.2:1026/NGSI10/updateContext* es el siguiente:

```
<?xml version="1.0" encoding="UTF-8"?>
<updateContextRequest>
  <contextElementList>
    <contextElement>
```

```
            <entityId type="UPCT:SENSOR" isPattern="false">
              <id>UPCT:TEMPERATURE:SENSOR</id>
            </entityId>
            <contextAttributeList>
              <contextAttribute>
                <name>temperature</name>
                <type>centigrade</type>
                <contextValue>25</contextValue>
              </contextAttribute>
              <contextAttribute>
                <name>humidity</name>
                <type>relative</type>
                <contextValue>60</contextValue>
              </contextAttribute>
            </contextAttributeList>
          </contextElement>
        </contextElementList>
        <updateAction>APPEND</updateAction>
      </updateContextRequest>
```

La respuesta del servidor, si se ha realizado satisfactoriamente la petición será la siguiente:

```
      <?xml version="1.0"?>
      <updateContextResponse>
        <contextResponseList>
          <contextElementResponse>
            <contextElement>
              <entityId type="UPCT:SENSOR" isPattern="false">
                <id>UPCT:TEMPERATURE:SENSOR</id>
              </entityId>
              <contextAttributeList>
                <contextAttribute>
                  <name>temperature</name>
                  <type>centigrade</type>
                  <contextValue/>
                </contextAttribute>
                <contextAttribute>
                  <name>humidity</name>
                  <type>relative</type>
                  <contextValue/>
                </contextAttribute>
              </contextAttributeList>
            </contextElement>
            <statusCode>
              <code>200</code>
              <reasonPhrase>OK</reasonPhrase>
            </statusCode>
          </contextElementResponse>
        </contextResponseList>
      </updateContextResponse>
```

Es posible eliminar atributos de una entidad con la misma petición y cambiando el contenido de la etiqueta <*updateAction*> por *DELETE*.

Cabe además la posibilidad de crear atributos metadata, que dan información del atributo al que se asocian. Algunos de esos metadata tienen características especiales, como es el caso de *location*. De la misma manera será posible modificar o consultar esas etiquetas.

De la misma manera, es posible guardar vectores de atributos en lugar de atributos únicos. Ello permite guardar mayor cantidad de valores en una entidad. En el caso de utilizar ficheros XML, se debe emplear la etiqueta <vector>. Si se utiliza JSON se hace más sencilla esta tarea, ya que sólo se debe introducir el vector tras el valor *attribute:*

I.3.2 Actualización de una entidad. Update context.

Siguiendo con el ejemplo, se van a actualizar los valores de contexto de los atributos *temperature* y *humidity* a 30 ºC y 70 %, respectivamente. Para ello, al igual que en el caso anterior, el formulario XML necesario debe ser enviado a la dirección *130.206.83.2:1026/NGSI10/updateContext* y el contenido del mismo sería el siguiente:

```
<?xml version="1.0" encoding="UTF-8"?>
<updateContextRequest>
  <contextElementList>
    <contextElement>
      <entityId type="UPCT:SENSOR" isPattern="false">
        <id>UPCT:TEMPERATURE:SENSOR</id>
      </entityId>
      <contextAttributeList>
        <contextAttribute>
          <name>temperature</name>
          <type>centigrade</type>
          <contextValue>30</contextValue>
        </contextAttribute>
        <contextAttribute>
          <name>humidity</name>
          <type>relative</type>
          <contextValue>70</contextValue>
        </contextAttribute>
      </contextAttributeList>
    </contextElement>
  </contextElementList>
  <updateAction>UPDATE</updateAction>
</updateContextRequest>
```

La respuesta del servidor, si se ha realizado satisfactoriamente la petición será:

```
<?xml version="1.0"?>
<updateContextResponse>
  <contextResponseList>
    <contextElementResponse>
      <contextElement>
        <entityId type="UPCT:SENSOR" isPattern="false">
          <id>UPCT:TEMPERATURE:SENSOR</id>
        </entityId>
        <contextAttributeList>
          <contextAttribute>
            <name>temperature</name>
            <type>centigrade</type>
            <contextValue/>
          </contextAttribute>
          <contextAttribute>
            <name>humidity</name>
            <type>relative</type>
            <contextValue/>
          </contextAttribute>
        </contextAttributeList>
      </contextElement>
      <statusCode>
        <code>200</code>
        <reasonPhrase>OK</reasonPhrase>
      </statusCode>
    </contextElementResponse>
  </contextResponseList>
</updateContextResponse>
```

Es posible actualizar solamente un atributo de la entidad eliminando las etiquetas correspondientes.

I.3.3 Consulta de una entidad. Query context.

Tal y como se ha comentado anteriormente, para consultar la entidad y obtener información de contexto de ella debe enviarse el formulario a la dirección *130.206.83.2:1026/NGSI10/queryContext*.

Existen varias formas de realizar una *query*. Una es pidiendo a la entidad toda la información que contiene mediante el formulario siguiente:

```
<?xml version="1.0" encoding="UTF-8"?>
<queryContextRequest>
  <entityIdList>
    <entityId type="UPCT:SENSOR" isPattern="false">
      <id>UPCT:TEMPERATURE:SENSOR</id>
    </entityId>
  </entityIdList>
  <attributeList/>
</queryContextRequest>
```

La otra es pidiendo específicamente qué atributos consultar. En este caso y siguiendo el ejemplo planteado, se puede consultar únicamente el valor de temperatura mediante el formulario siguiente:

```
<?xml version="1.0" encoding="UTF-8"?>
<queryContextRequest>
  <entityIdList>
    <entityId type="UPCT:SENSOR" isPattern="false">
      <id>UPCT:TEMPERATURE:SENSOR</id>
    </entityId>
  </entityIdList>
  <attributeList>
  <attribute>temperature</attribute>
  </attributeList>
</queryContextRequest>
```

Es también posible consultar el mismo atributo de varias entidades distintas añadiendo dentro de la etiqueta <*entityIdList*> otra <*entityId*> además de la existente.

En cualquier caso, la respuesta del servidor es similar a la siguiente, apareciendo más o menos atributos y entidades en la respuesta:

```
<?xml version="1.0"?>
<queryContextResponse>
  <contextResponseList>
    <contextElementResponse>
      <contextElement>
        <entityId type="UPCT:SENSOR" isPattern="false">
          <id>UPCT:TEMPERATURE:SENSOR</id>
        </entityId>
        <contextAttributeList>
          <contextAttribute>
            <name>temperature</name>
            <type>centigrade</type>
            <contextValue>30</contextValue>
          </contextAttribute>
          <contextAttribute>
            <name>humidity</name>
            <type>relative</type>
            <contextValue>70</contextValue>
          </contextAttribute>
        </contextAttributeList>
```

```
            </contextElement>
            <statusCode>
              <code>200</code>
              <reasonPhrase>OK</reasonPhrase>
            </statusCode>
          </contextElementResponse>
        </contextResponseList>
      </queryContextResponse>
```

Para consultar todas las entidades creadas de un tipo de sensor en una instancia, se puede utilizar como *id* de la entidad el valor <*id*>.*</*id*>. Además para hacer posible esta petición el valor *isPattern* debe ser *true*.

I.3.4 Suscripción de entidades. Subscribe context.

Las suscripciones se emplean para que una instancia informe a otra cuando un evento determinado ocurra. Cuando éste ocurre en la instancia 1, se crea o actualiza en la instancia 2 la misma entidad de forma que siempre permanecen sincronizadas.

El evento que origina ese intercambio de información puede ser de dos tipos *ONTIMEINTERVAL* y *ONCHANGE*. Cuando se utiliza una suscripción *ONINTERVAL* se producirá la actualización en periodos de tiempo determinados. En el caso de *ONCHANGE* se producirá cuando cambie el valor de contexto de algún atributo. También es posible seleccionar la duración de la suscripción mediante la etiqueta <*duration*>.

Existe además la etiqueta <*throttling*> que delimita el tiempo mínimo que debe transcurrir entre dos notificaciones. Ello hace que si se realizan muchas notificaciones de forma continuada, sólo se reciba una por cada intervalo de tiempo determinado por el valor dentro de la etiqueta

Continuando con el ejemplo, si se quisiera suscribir una nueva entidad llamada *NEW_ENTITY (130.206.83.3:1026)* a la instancia actual *UPCT (130.206.83.2:1026)* que además sea de tipo *ONCHANGE* se enviaría el siguiente formulario a la dirección *130.206.83.2:1026/NGSI10/subscribeContext*:

```
<?xml version="1.0"?>
<subscribeContextRequest>
  <entityIdList>
    <entityId type="UPCT:SENSOR" isPattern="false">
      <id>UPCT:TEMPERATURE:SENSOR</id>
    </entityId>
  </entityIdList>
  <reference>http://130.206.83.3:1026/accumulate</reference>
  <duration>P1M</duration>
  <notifyConditions>
    <notifyCondition>
      <type>ONCHANGE</type>
      <condValueList>
        <condValue>temperature</condValue>
      </condValueList>
    </notifyCondition>
  </notifyConditions>
  <throttling>PT5S</throttling>
</subscribeContextRequest>
```

En la respuesta del sistema, se aporta un identificador de la suscripción para tener acceso a ella posteriormente como a continuación se muestra:

```
<?xml version="1.0"?>
<subscribeContextResponse>
  <subscribeResponse>
    <subscriptionId>51c04a21d714fb3b37d7d5a7</subscriptionId>
    <duration>P1M</duration>
  </subscribeResponse>
</subscribeContextResponse>
```

I.4 Herramientas para la comunicación con FI-WARE.

A continuación se describen una serie de herramientas que simplifican el trabajo con la plataforma.

Es importante señalar que aunque algunas de las herramientas mostradas son válidas para cualquier sistema operativo, otras están disponibles únicamente para *Windows*.

Para poder interactuar con FI-WARE es necesario utilizar un cliente HTTP como ya se ha mencionado en varias ocasiones. Sin embargo, una de las grandes virtudes de las instancias de FI-WARE es que permiten la conexión por SSH siempre que se configuren correctamente.

Dicha conexión se puede realizar desde un sistema operativo basado en Linux mediante el comando

```
ssh –i keypair.pem root@ip (ej: ssh –i enrique_keyapair.pem root@130.206.83.2)
```

En el caso de utilizar *Windows*, hay herramientas como *Putty,* que permiten varios tipos de conexiones, entre ellas, SSH.

I.4.1 Putty. Conexión SSH en Windows.

Putty es una herramienta gratuita disponible para *Windows* que va a permitir la conexión remota con la instancia de FI-WARE.

Para utilizarla, en primer lugar debe convertirse la clave privada *keypair.pem* a una compatible *keypair.ppk*. Para ello se utiliza la aplicación *Puttygen*.

Se ejecuta la herramienta *Putty* y se configuran los parámetros *Hostname* y *port* como se muestra en la Ilustración I.8.

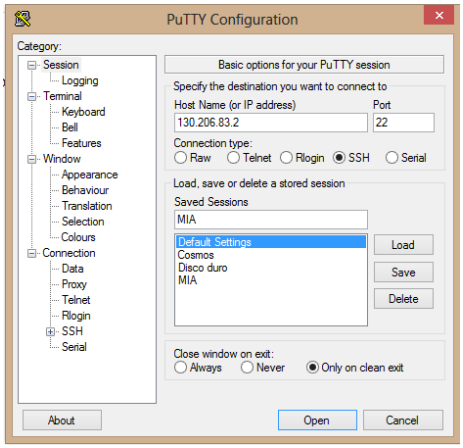

Ilustración I.8. Vista de la herramienta Putty.

Finalmente, se selecciona el archivo *keypair.ppk* desde la llave *Connection/SSH/Auth*.

Haciendo click en el botón *Open* se inicia la conexión y el servidor preguntará por el nombre de usuario. De forma general se debe seleccionar *root* (ver Ilustración I.9)

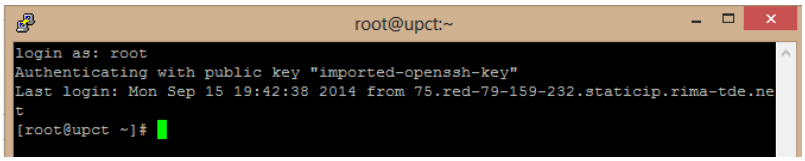

Ilustración I.9. Vista de la conexión SSH con Putty

I.4.2 cURL.

Una vez realizada la conexión SSH, es posible llevar a cabo todas las operaciones señaladas en el apartado I.3 de este anexo mediante una herramienta llamada *cURL*. Ésta permite transferir archivos con sintaxis *URL* desde la línea de comandos.

La sintaxis de esta herramienta es de la forma:

```
(curl localhost:1026/<operation_url> -s -S [headers]' -d @- )<<EOF
[payload]
EOF
```

Donde el contenido entre *<<EOF* y *EOF* será el formulario XML a enviar.

Según el ejemplo seguido a lo largo del capítulo, la sintaxis del comando curl para actualizar el valor de contexto de un atributo sería:

```
(curl localhost:1026/NGSI10/updateContext -s -S --header 'Content-Type:
application/xml' -d @- | xmllint --format - ) <<EOF
<?xml version="1.0" encoding="UTF-8"?>
<updateContextRequest>
  <contextElementList>
    <contextElement>
      <entityId type="UPCT:SENSOR" isPattern="false">
        <id>UPCT:TEMPERATURE:SENSOR</id>
```

```
            </entityId>
            <contextAttributeList>
              <contextAttribute>
                <name>temperature</name>
                <type>centigrade</type>
                <contextValue>30</contextValue>
              </contextAttribute>
            </contextAttributeList>
          </contextElement>
        </contextElementList>
        <updateAction>UPDATE</updateAction>
      </updateContextRequest>
      EOF
```

También es posible enviar desde un documento de texto el formulario XML mediante la sintaxis

```
curl localhost:1026/<operation_url> -s -S [headers]' -d @XML.txt
```

I.4.3 WinSCP. Transferencia remota de archivos en Windows.

WindSCP es una aplicación gratuita para el sistema operativo *Windows,* que permite la transferencia de archivos entre una máquina remota y el sistema local.

Para la conexión, esta aplicación demanda un usuario y contraseña, pero no permite la introducción de una clave privada desde un archivo. Por ello, antes de utilizarla es conveniente acceder a la máquina por SSH con otra aplicación como *putty* e introducir el comando *passwd* en la línea de comandos para generar otra clave que permita entrar en el sistema.

Su funcionamiento es muy básico, tras ejecutarla sólo será necesario introducir los parámetros básicos de la máquina para acceder a ella, visualizar todos sus directorios y archivos y transferirlos. La configuración para el ejemplo se muestra en la Ilustración I.10.

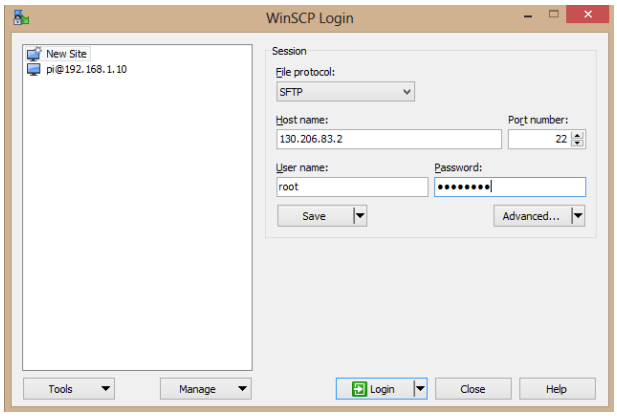

Ilustración I.10. Vista de la configuración de la herramienta WinSCP.

Una vez establecida la conexión la interfaz mostrará una ventana doble en la que aparece el equipo local a la izquierda y el servidor remoto a la derecha, de forma que la transferencia y visualización se hace de forma intuitiva (ver Ilustración I.11).

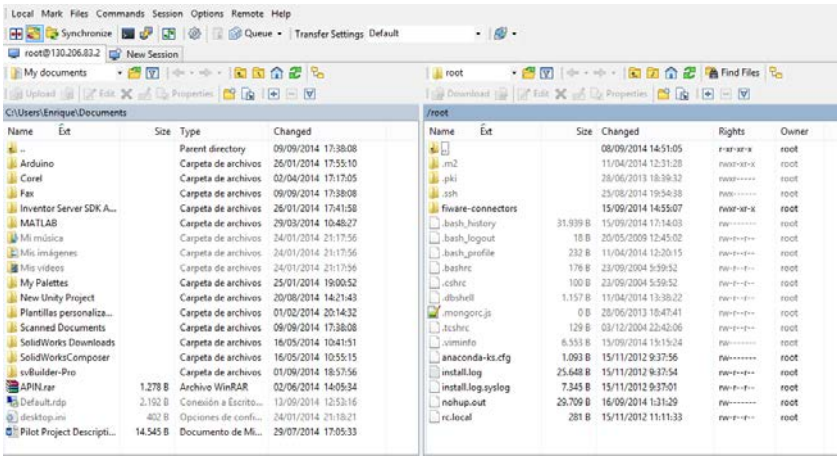

Ilustración I.10. Vista de la interfaz de la herramienta WinSCP.

I.5 Utilización de Cosmos. Configuración del conector Cygnus.

Para conseguir la persistencia de datos en Cosmos es necesario conocer el proceso que debe seguir la información.

En primer lugar, debe existir una instancia (la propia u otra cualquiera) que contenga una instalación del inyector *Cygnus*. Éste se encarga de enviar la información recibida a la máquina virtual donde reside *Cosmos* y a hacer que éstos persistan allí, es decir, inyecta la información en el histórico.

Esa información puede almacenarse en los formatos: *HDFS, CKAN* y *MySQL*.

A su vez *Cygnus* es un complemento del conector *Flume*, un conector que acumula información de fuentes reales y crea eventos. Es empleado para la generación de notificaciones de sistema. Concretamente, se utiliza en muchas plataformas como *Twitter* o el sistema de notificaciones de *Android*.

La forma de hacer que *Cygnus* reciba información es suscribirlo a la instancia que contiene dicha información, generalmente con una suscripción de tipo *ONCHANGE*.

Tras poner en marcha una instancia de FI-WARE como se ha detallado anteriormente (versión 0.14.0 de orion), ésta contiene una instalación de *Flume* en la ruta */usr/local*. Sin embargo, ésta requiere de la instalación del complemento externo *Cygnus* para que pueda interactuar con *Cosmos*.

I.5.1 Instalación del complemento Cygnus.

En primer lugar se debe establecer la conexión con la instancia por SSH, donde por defecto el directorio será /root/.

Se cambiará la ruta de trabajo a la de la instalación de *Flume*, mediante el comando:

```
cd ../usr/local/
```

Una vez allí se deben crear tres nuevos directorios en la carpeta de instalación de flume *apache-flume-1.4.0-bin/plugins.d.*

```
mkdir -p apache-flume-1.4.0-bin/plugins.d/cygnus/
mkdir apache-flume-1.4.0-bin/plugins.d/cygnus/lib
mkdir apache-flume-1.4.0-bin/plugins.d/cygnus/libext
```

La creación del directorio *plugins.d* está destinada a introducir software de terceros, como *Cygnus*.

Posteriormente, se debe descargar la librería *libthrift 0.9.1* en la ubicación *apache-flume-1.4.0-bin/bin.*

```
cd apache-flume-1.4.0-bin/bin
wget http://repo1.maven.org/maven2/org/apache/thrift/libthrift/0.9.1/libthrift-0.9.1.jar
```

A continuación, se procede con la descarga de *Cygnus.* Para ello, es necesario instalar en la máquina remota una herramienta capaz de descargar archivos desde directorios *Git Hub,* mediante el comando:

```
sudo yum install git
```

A continuación se realiza la descarga del paquete *fiware-connectors,* donde reside *Cygnus.*

```
git clone https://github.com/telefonicaid/fiware-connectors.git
```

Se cambia la ruta de trabajo al nuevo directorio descargado, y se selecciona la versión de *Cygnus* que se desea instalar, en este caso, la 0.3.

```
cd fiware-connectors
git checkout release/0.3
```

A continuación se vuelve a cambiar el directorio de trabajo a *fiware-connectors/flume,* y se compila el archivo *mvn* en la carpeta de instalación de *Maven.*

```
cd fiware-connectors/flume
../../apache-maven-3.2.1/bin/mvn clean compile assembly:single
```

Por último, se copia el *.jar* de *Cygnus* en la carpeta creada anteriormente *plugins.d* de la instalación de flume:

```
cp target/cygnus-0.3-jar-with-dependencies.jar ../../apache-flume-1.4.0-
bin/plugins.d/cygnus/lib
```

Cygnus incluye un archivo de configuración, que se encuentra en la dirección *fiware-connectors/flume/conf/cygnus.conf.template*. Éste debe reubicarse en la carpeta *apache-flume-1.4.0-bin/conf/* y renombrarse *cygnus.conf*. Antes de ello se vuelve a cambiar el directorio de trabajo a */usr/local/*:

```
cd ../../

cp fiware-connectors/flume/conf/cygnus.conf.template apache-flume-1.4.0-bin/conf/

cd apache-flume-1.4.0-bin/conf/

mv cygnus.conf.template cygnus.conf
```

Para cambiar la configuración del sistema se edita dicho archivo mediante el editor de texto *vim*.

```
vim cygnus.conf
```

La configuración para el caso estudiado a lo largo del capítulo es la siguiente:

```
# APACHE_FLUME_HOME/conf/cygnus.conf

# The next tree fields set the sources, sinks and channels used by Cygnus. You could use
different names than the
# ones suggested below, but in that case make sure you keep coherence in properties
names along the configuration file.
# Regarding sinks, you can use multiple ones at the same time; the only requirement is
to provide a channel for each
# one of them (this example shows how to configure 3 sinks at the same time).
cygnusagent.sources = http-source
cygnusagent.sinks = hdfs-sink
cygnusagent.channels = hdfs-channel

#============================================
# source configuration
# channel name where to write the notification events
cygnusagent.sources.http-source.channels = hdfs-channel mysql-channel ckan-channel
# source class, must not be changed
cygnusagent.sources.http-source.type = org.apache.flume.source.http.HTTPSource
# listening port the Flume source will use for receiving incoming notifications
cygnusagent.sources.http-source.port = 5050
# Flume handler that will parse the notifications, must not be changed
cygnusagent.sources.http-source.handler =
es.tid.fiware.fiwareconnectors.cygnus.handlers.OrionRestHandler
# URL target
cygnusagent.sources.http-source.handler.notification_target = /notify
# Default organization (organization semantic depend on the persistence sink)
cygnusagent.sources.http-source.handler.default_organization = org42

# ============================================
# OrionHDFSSink configuration
# channel name from where to read notification events
cygnusagent.sinks.hdfs-sink.channel = hdfs-channel
# sink class, must not be changed
```

```
cygnusagent.sinks.hdfs-sink.type =
es.tid.fiware.fiwareconnectors.cygnus.sinks.OrionHDFSSink
# The FQDN/IP address of the Cosmos deployment where the notification events will be
persisted
cygnusagent.sinks.hdfs-sink.cosmos_host = 130.206.80.46
# port of the Cosmos service listening for persistence operations; 14000 for httpfs,
50070 for webhdfs and free choice for inifinty
cygnusagent.sinks.hdfs-sink.cosmos_port = 14000
# default username allowed to write in HDFS
cygnusagent.sinks.hdfs-sink.cosmos_default_username = quiquehz
# default password for the default username
cygnusagent.sinks.hdfs-sink.cosmos_default_password = 1442200106
# HDFS backend type (webhdfs, httpfs or infinity)
cygnusagent.sinks.hdfs-sink.hdfs_api = httpfs
# how the attributes are stored, either per row either per column (row, column)
cygnusagent.sinks.hdfs-sink.attr_persistence = column
# prefix for the database and table names, empty if no prefix is desired
cygnusagent.sinks.hdfs-sink.naming_prefix =
# Hive port for Hive external table provisioning
cygnusagent.sinks.hdfs-sink.hive_port = 10000

#==========================================
# hdfs-channel configuration
# channel type (must not be changed)
cygnusagent.channels.hdfs-channel.type = memory
# capacity of the channel
cygnusagent.channels.hdfs-channel.capacity = 1000
# amount of bytes that can be sent per transaction
cygnusagent.channels.hdfs-channel.transactionCapacity = 100
```

La cual está orientada a una persistencia de datos en archivos de texto en sistema de archivs de *Hadoop (HDFS).*

I.5.2 Acceso a la máquina virtual de Cosmos.

Es necesario tener permisos de la máquina de *Cosmos* para poder almacenar información en ella. Para conseguir dichos permisos se debe acceder a la web http://cosmos.lab.fi-ware.org/cosmos-gui/.

Por seguridad, dicha máquina sólo es accesible desde máquinas de FI-WARE por lo que la forma de conectarse a ella por SSH es a través de la máquina instanciada por el usuario.

Una vez en la ventana de comandos de la instancia de FI-WARE, se accede por SSH a la de Cosmos mediante la introducción del comando

```
ssh user@cosmos.lab.fi-ware.org

   ó

ssh user@130.206.80.46
```

Para la visualización de la información en la línea de comandos se deben utilizar comandos de propios del sistema de archivos de *Hadoop.*

Para ver los directorios existentes se emplea el comando:

```
hadoop fs –du /user/quiquehz/org42
```

Para obtener un listado de los archivos de un directorio:

```
hadoop fs –ls /user/quiquehz/org42
```

Para abrir un determinado archivo:

```
hadoop fs –cat /user/quiquehz/org42/file.txt
```

Anexo II

Detalle del software desarrollado para el caso de estudio.

II.1 Introducción.

En este anexo se describe con mayor profundidad el software desarrollado para implementar el parking inteligente descrito en el Capítulo 4. En primer lugar se detallará la conexión individual de los diferentes sensores utilizados en los dispositivos. Posteriormente, se expondrá el código completo desarrollado para la implementación de los dispositivos del parking, describiendo las sentencias y funciones de mayor relevancia, que no fueron descritas en el Capítulo 4.

El código completo de éste caso de estudio se puede encontrar en https://github.com/HenryHZ/UPCT-SMART-PARKING y en el CD adjunto en el presente proyecto.

II.2 Descripción de la conexión de sensores con los dispositivos empotrados.

A continuación se detalla el software y hardware necesarios para la realización de las conexiones de los sensores.

II.2.1 Implementación del sensor HC-SR501 en Arduino.

El sensor PIR HC-SR501 obtiene la alimentación necesaria para funcionar de los pines *VCC* y *GND* y envía un pulso de nivel de voltaje alto (5V) cuando detecta un movimiento. La conexión con Arduino es posible realizarla como explica la Tabla II.1

Elemento	Pin elemento	Pin Arduino
HC-SR501	VCC	5 V
	Output	Digital 2
	GND	GND

Tabla II.1. Conexión del sensor HC-SR501 con Arduino.

En este caso el Sketch que permite utilizar dicho sensor se basa en una interrupción, de forma que cuando se detecta movimiento y Arduino detecta un flanco de subida, directamente se realiza la acción correspondiente.

Para ello se utiliza la función *attachInterrupt(),* donde se incluyen los argumentos correspondientes al pin que recibe la detección (0 si es el pin 2 y 1 si es el pin 3), la función que ejecuta y el tipo de detección (*RISING,* flanco ascendente). En el código se ha incluido el envío del mensaje "objeto detectado" cuando se realice la detección. El Sketch resultante es el siguiente:

```
boolean detectado;

void setup()
{
  Serial.begin(9600);
  detectado = false;
  attachInterrupt(0, disparo, RISING);
}

void loop()
{
  if (detectado){
    Serial.println("Objeto detectado");
    detectado = false;
    attachInterrupt(0, disparo, RISING);
  }
}

void disparo()
{
  if (!detectado){
    detectado = true;
    detachInterrupt(0);
  }
}
```

II.2.2 Implementación del sensor SHT11 en Arduino.

El sensor SHT11 funciona mediante una conexión I^2C con Arduino y es necesario enviarle una serie de comandos en codificación binaria para que realice las diferentes

acciones. Todo ello se encuentra desarrollado en la librería de *Sensirión,* disponible para la serie de sensores SHT1x.

La conexión realizada con Arduino se muestra en la Tabla II.2, donde se puede apreciar que el sensor tiene 4 pines de salida, 2 de alimentación y 2 para señal de reloj y datos.

Elemento	Pin elemento	Pin Arduino
SHT11	VCC	5V
	CLK	Digital 3
	DATA	Digital 2
	GND	GND

Tabla II.2. Conexión del sensor SHT11 con Arduino.

La librería señalada pone a disposición del usuario una serie de funciones que permiten la comunicación con el sensor. Para la realización de mediciones son necesarias dos sentencias: la inicialización de los pines de datos y reloj con *Sensirion tempSensor =Sensirion()* y la propia toma de datos con la función *tempSensor.measure(),* señalando las variables donde se guardará el resultado. A continuación se muestra un ejemplo de sketch que muestra las mediciones a través del monitor serial.

```
#include <Sensirion.h>

const uint8_t dataPin  =  2;
const uint8_t clockPin =  3;

float temperature;
float humidity;
float dewpoint;

Sensirion tempSensor = Sensirion(dataPin, clockPin);

void setup()
{
  Serial.begin(9600);
}

void loop()
{
  tempSensor.measure(&temperature, &humidity, &dewpoint);

  Serial.print("Temperature: ");
  Serial.print(temperature);

  Serial.print(" C, Humidity: ");
  Serial.print(humidity);

  Serial.print(" %, Dewpoint: ");
  Serial.print(dewpoint);
  Serial.println(" C");

  delay(5000);
}
```

II.2.3 Implementación del sensor SRF02 en Arduino.

El sensor SRF02 se comunica con Arduino usando el bus I^2C, al igual que ocurría con el SHT11. Sin embargo, en este caso no existe una librería para su utilización, por lo que se ha creado un Sketch que permite la obtención de datos mediante el envío de los comandos binarios necesarios con ayuda de la librería de Arduino <Wire.h>, que permite la conexión I^2C. Este sketch pasará a ser una función propia llamada *getDistance()* en la implementación final del caso práctico [12].

La conexión física entre los sensores se muestra en la Tabla II.3.

Elemento	Pin elemento	Pin Arduino
SRF02	1	5V
	2	Analog 4
	3	Analog 5
	5	GND

Tabla II.3. Conexión del sensor SFR02 con Arduino

La librería necesita ser inicializada como todas las demás, en este caso, mediante la línea *#include <Wire.h>*. Además, la transmisión entre sensor y Arduino debe inicializarse mediante *Wire.begin()*.

```
#include <Wire.h>

void setup()
{
  Wire.begin();
  Serial.begin(9600);
}
```

Se inicializa la variable en la que se guardarán las medidas de distancia *reding* y se inicia el primer paso de la lectura, que es la preparación del sensor, indicando la dirección de donde se obtendrán los datos mediante *Wire.beginTransmision(112)*, inicializando el puntero de registro a cero con *Write.write()*. Finalmente, se selecciona en qué unidades se obtienen las medidas (0x51 centimetros, 0x50 pulgadas).

```
int reading = 0;

void loop()
{
  Wire.beginTransmission(112);
  Wire.write(byte(0x00));
  Wire.write(byte(0x51));
  Wire.endTransmission();
```

La hoja de características detalla que es necesario esperar al menos 65 milisegundos para comenzar a medir.

```
  delay(70);
```

A continuación, se prepara el sensor para poder devolver una respuesta al dispositivo, para lo que se selecciona el puntero del registro a 1 (0x02)

```
    Wire.beginTransmission(112);
    Wire.write(byte(0x02));
    Wire.endTransmission();
```

El sensor devolverá dos bytes de datos, que el sistema debe recibir mediante *Wire.requestFrom(112,2),* donde 2 indica el número de bytes.

```
    Wire.requestFrom(112, 2);
```

Finalmente, se comprueba que realmente se han recibido dos bytes, se recibe la información como cadena de 16 bits que se deben separar en 2, para lo cual se cortan a los 8 bits. Finalmente, se imprime por pantalla el resultado.

```
    if(2 <= Wire.available())
    {
      reading = Wire.read();
      reading = reading << 8;
      reading |= Wire.read();
      Serial.println(reading);
    }
    delay(250);
}
```

II.2.4 Implementación del sensor HC-SR04 en Raspberry Pi.

Existen varios lenguajes de programación válidos para generar aplicaciones para Raspberry Pi. En este caso se utiliza C++. Para la implementación del sensor HC-SR04, se utiliza la librería *lbcm2835,* ya que simplifica el uso del GPIO de Raspberry Pi.

La conexión física empleada se muestra en la Tabla II.4.

Elemento	Pin elemento	Pin RPi	GPIO
HC-SR04	VCC	Pin 2	
	Trig	Pin 11	GPIO 17
	Echo	Pin 16	GPIO 23
	GND	GND	

Tabla II.4. Conexión del sensor HC-SR04 con Rasbperry Pi.

En el software desarrollado, en primer lugar se incluyen las librerías necesarias y se inicializan los pines GPIO que van a ser utilizados. La librería empleada permite dicha inicialización mediante el comando *#define NOMBRE_PIN RPI_GPIO_NºPIN*. También se selecciona la frecuencia de funcionamiento del GPIO con *BCM2835_CORE_CLK*.

```
#include <bcm2835.h>
#include <stdio.h>
```

```
#define BCM2835_CORE_CLK_HZ 250000000

#define PIN_TRIG RPI_GPIO_P1_11
#define PIN_ECHO RPI_GPIO_P1_16
```

Si es posible iniciar los pines señalados se continua con el programa. Se selecciona si los pines son de entrada o salida mediante el comando *bcm2835_gpio_fsel* .

```
int main(int argc, char **argv)
{
    if (!bcm2835_init()){
        return 1;
       printf("FIN");
    }
    bcm2835_gpio_fsel(PIN_TRIG, BCM2835_GPIO_FSEL_OUTP);
    bcm2835_gpio_fsel(PIN_ECHO, BCM2835_GPIO_FSEL_INPT);
    delay(500);
```

El funcionamiento del sensor indicado por su hoja de características es el siguiente:

1. Se envía un pulso de duración 10 μs al disparador del sensor.

2. El sensor envía un pulso de ultrasonidos de frecuencia 40 Hz y lo recibe.

3. Desde que se envía el pulso hasta que se recibe, un valor de tensión alto por el pin *echo* es recibido.

Por lo tanto, la forma de medir la distancia es contabilizando el tiempo que tarda en enviar el pulso y volver. Para tal fin, se inicializa un contador (*contador),* que medirá el tiempo que transcurre desde el envío del pulso hasta que es recibido, el tiempo de ida y vuelta.

En primer lugar se envía el pulso de 10 μs, que hará que comience a medir el sensor y se inicializa el contador.

```
while(1){
 bcm2835_gpio_write(PIN_TRIG, HIGH);
       delay(0.01);
 bcm2835_gpio_write(PIN_TRIG, LOW);
       float contador=0;
```

El sistema queda a la espera de que se comience a medir.

```
while(!bcm2835_gpio_lev(PIN_ECHO)){
}
```

Cuando el pulso es enviado comienza otro bucle while, en el que cada iteración se distancia de la siguiente 50 μs. El contador mide cuantas iteraciones transcurren, de forma que al final es posible saber el tiempo de ida y vuelta mediante la operación *contador**$50 \cdot 10^{-3}$s.

```
while(bcm2835_gpio_lev(PIN_ECHO)){
bcm2835_delayMicroseconds(50);
contador++;
}
```

Una vez recibido el pulso, el pin echo deja de enviar la señal de 5 V y el contador se detiene. Para obtener la distancia se multiplica el tiempo de ida y vuelta por la velocidad del sonido. Esa la distancia es la de ida y vuelta, por lo tanto, se debe dividir entre dos para obtener la distancia real.

```
float distance =(contador)*0.00005*340/2;
```

Mediante varias sentencias *if* se comprueba si la distancia es mayor o menor de 1 metro, y se muestra el resultado en el monitor serial en las unidades más oportunas.

```
if(distance>1.0){
printf("Distancia: %f metros\n", distance);
}
if(distance<1.0){
printf("Distancia: %f centimetros\n", distance*100);
}
```

Finalmente, se establece un *delay()* que va a marcar el tiempo entre medidas y se cierran los pines del GPIO.

```
            delay(1000);
            }
        bcm2835_close();
        return 0;
    }
```

II.2.5 Implementación del sensor SHT11 en Raspberry Pi.

El sensor SHT11 emplea una comunicación I^2C, al igual que otros de los sensores estudiados, y por tanto, se desarrollará el software necesario para enviar los comandos adecuados. En cuanto al hardware, se conecta como se expone en la Tabla II.5 [13].

Elemento	Pin elemento	Pin RPi	GPIO
SHT11	VCC	Pin 1	
	CLK	Pin 15	GPIO 22
	DATA	Pin 22	GPIO 25
	GND	GND	
	Resistencia 10 kΩ DATA-VCC		

Tabla II.5. Conexión del sensor SHT11 con Rasbperry Pi.

La implementación sensor SHT11 se lleva a cabo en varios archivos de C++ para facilitar la tarea.

- RPi_SHT1x.h, es el archivo que contiene la inicialización para GPIO de los comandos a enviar y la inicialización de las principales funciones del sistema.

- RPi_SHT1x.cpp contiene el desarrollo de las funciones anteriormente citadas.

- testSHT11.cpp es el archivo principal que contiene la ejecución del programa que efectúa las medidas utilizando todo lo desarrollado en los otros dos archivos.

Toda la documentación y código relativos a la conexión de este sensor ha sido obtenida de la web https://www.john.geek.nz/2012/08/reading-data-from-a-sensirion-sht1x-with-a-raspberry-pi/

El contenido del archivo *testSHT1x.cpp* se utiliza para generar las funciones *getTemp() y getHumid()* en el archivo *disp4.cpp,* creado para la implementación del dispositivo 4 del caso de estudio.

El contenido del archivo RPi_SHT1x.h comienza con la selección de los pines del GPIO que utilizará el sensor, la dirección de éstos (Input/Output) y los comandos binarios que el dispositivo utiliza. Finalmente, inicializa todas las funciones que serán utilizadas.

```
        #ifndef RPI_SHT1x_H_
        #define RPI_SHT1x_H_

        #include <bcm2835.h>
        #include <unistd.h>
        #include <stdio.h>

        #define TRUE    1
        #define FALSE   0
        #define SHT1x_DELAY delayMicroseconds(2)

        #define RPI_GPIO_SHT1x_SCK RPI_GPIO_P1_15
        #define RPI_GPIO_SHT1x_DATA RPI_GPIO_P1_22

        #define SHT1x_SCK_LO  bcm2835_gpio_write(RPI_GPIO_SHT1x_SCK, LOW)
        #define SHT1x_SCK_HI  bcm2835_gpio_write(RPI_GPIO_SHT1x_SCK, HIGH)
        #define SHT1x_DATA_LO bcm2835_gpio_write(RPI_GPIO_SHT1x_DATA, LOW);
         bcm2835_gpio_fsel(RPI_GPIO_SHT1x_DATA, BCM2835_GPIO_FSEL_OUTP)
        #define SHT1x_DATA_HI  bcm2835_gpio_fsel(RPI_GPIO_SHT1x_DATA,BCM2835_
        GPIO_FSEL_INPT)
        #define SHT1x_GET_BIT bcm2835_gpio_lev(RPI_GPIO_SHT1x_DATA)

        #define SHT1x_MEAS_T  0x03          // Start measuring of temperature.
        #define SHT1x_MEAS_RH 0x05          // Start measuring of humidity.
        #define SHT1x_STATUS_R0x07          // Read status register.
        #define SHT1x_STATUS_W0x06          // Write status register.
        #define SHT1x_RESET   0x1E          // Perform a sensor soft reset.


/* Public Functions ------------------------------------------------------- */
        void SHT1x_Transmission_Start( void );
        unsigned char SHT1x_Readbyte( unsigned char sendAck );
        unsigned char SHT1x_Sendbyte( unsigned char value );
        void SHT1x_InitPins( void );
        unsigned char SHT1x_Measure_Start( SHT1xMeasureType type );
        unsigned char SHT1x_Get_Measure_Value(unsigned short int * value );
        void SHT1x_Reset();
        unsigned char SHT1x_Mirrorbyte(unsigned char value);
        void SHT1x_Xrc_check(unsigned char value);
        void SHT1x_Calc(float *p_humidity ,float *p_temperature);
        #endif
```

El contenido del archivo RPi_SHT1x.cpp se muestra dividido en las distintas funciones que lo conforman.

La primera de ellas es *SHT1x_InitPins,* la cual inicializa los pines GPIO.

```
void SHT1x_InitPins( void ) {
        bcm2835_gpio_write(RPI_GPIO_SHT1x_SCK, LOW);
        bcm2835_gpio_fsel(RPI_GPIO_SHT1x_SCK, BCM2835_GPIO_FSEL_OUTP);
        bcm2835_gpio_write(RPI_GPIO_SHT1x_SCK, LOW);
        bcm2835_gpio_set_pud(RPI_GPIO_SHT1x_DATA, BCM2835_GPIO_PUD_OFF);
        bcm2835_gpio_write(RPI_GPIO_SHT1x_DATA,LOW);
        bcm2835_gpio_fsel(RPI_GPIO_SHT1x_DATA, BCM2835_GPIO_FSEL_OUTP);
        bcm2835_gpio_set_pud(RPI_GPIO_SHT1x_DATA, BCM2835_GPIO_PUD_OFF);
        bcm2835_gpio_write(RPI_GPIO_SHT1x_DATA,LOW);
}
```

La función *SHT1x_reset()* realiza un soft reset del sensor, el cual está programado en él y es accesible mediante el comando *0x1E*.

```
void SHT1x_Reset() {
        unsigned char i;
        SHT1x_DATA_HI;
        SHT1x_DELAY;
        for (i=9; i; i--)
        {
                SHT1x_SCK_HI;  SHT1x_DELAY;
                SHT1x_SCK_LO;  SHT1x_DELAY;
        }
        SHT1x_Transmission_Start();
        SHT1x_Sendbyte(SHT1x_RESET);  // Soft reset
}
```

Cuando se realiza una transmisión al sensor se deben tener en cuenta unos tiempos de espera o *delays*. La función *SHT1x_Transmission_Start()* se encarga de ello.

```
void SHT1x_Transmission_Start( void ) {
        SHT1x_SCK_HI;  SHT1x_DELAY;
        SHT1x_DATA_LO; SHT1x_DELAY;
        SHT1x_SCK_LO;  SHT1x_DELAY;
        SHT1x_SCK_HI;  SHT1x_DELAY;
        SHT1x_DATA_HI; SHT1x_DELAY;
        SHT1x_SCK_LO;  SHT1x_DELAY;
        SHT1x_crc = SHT1x_Mirrorbyte( SHT1x_status_reg & 0x0F );
}
```

Para el envío de la información se crea la función *SHT1x_sendByte()*.

```
unsigned char SHT1x_Sendbyte( unsigned char value ) {
        unsigned char mask;
        unsigned char ack;

        for(mask = 0x80; mask; mask>>=1){
                SHT1x_SCK_LO;  SHT1x_DELAY;

                if( value & mask ){
                        SHT1x_DATA_HI;        SHT1x_DELAY;
                }
                else{
                        SHT1x_DATA_LO; SHT1x_DELAY;
                }

                SHT1x_SCK_HI;  SHT1x_DELAY;
        }
        SHT1x_SCK_LO;  SHT1x_DELAY;
        SHT1x_DATA_HI; SHT1x_DELAY;
        SHT1x_SCK_HI;  SHT1x_DELAY;

        ack = 0;

        if(!SHT1x_GET_BIT)
        ack = 1;
```

```
        SHT1x_SCK_LO;  SHT1x_DELAY;

        SHT1x_Crc_Check(value);

        return ack;
}
```

Análogamente, para la recepción de información se utiliza *SHT1x_readByte()*.

```
unsigned char SHT1x_Readbyte( unsigned char send_ack ) {
        unsigned char mask;
        unsigned char value = 0;

        for(mask=0x80; mask; mask >>= 1 ){
                SHT1x_SCK_HI;  SHT1x_DELAY;
                if( SHT1x_GET_BIT != 0 )
                        value |= mask;

                SHT1x_SCK_LO;  SHT1x_DELAY;
        }

        if ( send_ack ){
                SHT1x_DATA_LO; SHT1x_DELAY;
        }

        SHT1x_SCK_HI;  SHT1x_DELAY;
        SHT1x_SCK_LO;  SHT1x_DELAY;

        if ( send_ack ){
                SHT1x_DATA_HI; SHT1x_DELAY;
        }
        return value;
}
```

La función *SHT1x_Measure_Start()* se encarga de iniciar la medición de temperatura y humedad relativa.

```
unsigned char SHT1x_Measure_Start( SHT1xMeasureType type ) {
        SHT1x_Transmission_Start();
        return SHT1x_Sendbyte( (unsigned char) type );
}
```

SHT1x_Get_Measure() obtiene el valor de la medición iniciada anteriormente.

```
unsigned char SHT1x_Get_Measure_Value( unsigned short int * value ) {
        unsigned char * chPtr = (unsigned char*) value;
        unsigned char checksum;
        unsigned char delay_count=62;

        while( SHT1x_GET_BIT )
        {
                delayMicroseconds(5000);
                delay_count--;
                if (delay_count == 0)
                        return FALSE;
        }
        *(chPtr + 1) = SHT1x_Readbyte( TRUE );
        SHT1x_Crc_Check(*(chPtr + 1));

        *chPtr = SHT1x_Readbyte( TRUE );
        SHT1x_Crc_Check(* chPtr);
        checksum = SHT1x_Readbyte( FALSE );
        return SHT1x_Mirrorbyte( checksum ) == SHT1x_crc ? TRUE : FALSE;
}
```

Finalmente, el archivo *testSHT11.cpp* realiza una medición y la envía al monitor serial. Realiza además una comprobación de los pines y del estado del sistema de forma que si algo no funciona bien no se ejecutar dicha toma de datos.

```
#include <bcm2835.h>
#include <stdio.h>
#include "RPi_SHT1x.h"
#include <time.h>

void printTempAndHumidity(void){
        unsigned char noError = 1;
        value humi_val,temp_val;

        delay(20);

        SHT1x_InitPins();

        SHT1x_Reset();

        noError = SHT1x_Measure_Start( SHT1xMeaT );
        if (!noError) {
                return;
                }

        noError = SHT1x_Get_Measure_Value( (unsigned short int*) &temp_val.i );
        if (!noError) {
                return;
                }

        noError = SHT1x_Measure_Start( SHT1xMeaRh );
        if (!noError) {
                return;
                }

        noError = SHT1x_Get_Measure_Value( (unsigned short int*) &humi_val.i );
        if (!noError) {
                return;
                }

        temp_val.f = (float)temp_val.i;
        humi_val.f = (float)humi_val.i;

        SHT1x_Calc(&humi_val.f, &temp_val.f);

        printf("Temperature: %0.2f%cC\n",temp_val.f,0x00B0);
        printf("Humidity: %0.2f%%\n",humi_val.f);
}

int main (){
        if(!bcm2835_init())return 1;

        printTempAndHumidity();
        return 1;
}
```

II.3 Funciones desarrolladas.

En este apartado se exponen las funciones que han sido utilizadas en el desarrollo de los dispositivos del parking inteligente estudiado como caso práctico.

Aunque muchas de las funciones son comunes para los dispositivos Arduino y Raspberry Pi, las sentencias que las conforman serán, por lo general, distintas, por lo que se detallará el código para cada caso.

II.3.1 UpdateLed.

La función *updateLed()* es una función utilizada para cambiar el estado de los LEDs, para señalar si una plaza del parking está ocupada o no. No devuelve ningún valor y necesita un entero (integer) como argumento. Dicho entero será el que defina que led se enciende y qué led se apaga.

En el caso de Arduino, los pines donde están conectados se guardan en las variables *green_led* y *red_led*. Cuando el valor pasado como argumento es 0 el led verde se enciende y el rojo se apaga. Ocurre lo contrario si es 1.

```
void updateLed(int val){
  if(val<0){
    digitalWrite(red_led, LOW);
    digitalWrite(green_led, HIGH);
  }else{
    digitalWrite(red_led, HIGH);
    digitalWrite(green_led, LOW);
  }
}
```

El mismo código se utiliza para el dispositivo 1 que señaliza si hay plazas libres o no en el parking cambiando la condición de la sentencia *if* de 0 a *max_car*.

Para el caso de Raspberry Pi, el sistema es el mismo pero cambia la sintaxis para encender y apagar los pines. El led verde se encuentra conectado en el pin *GREEN_LED* del GPIO y el rojo en *RED_LED*. La función es la siguiente:

```
void updateLed(int value){
       if(value==0){
       bcm2835_gpio_write(RED_LED, LOW);
       bcm2835_gpio_write(GREEN_LED,HIGH);
       }else{
       bcm2835_gpio_write(RED_LED, HIGH);
       bcm2835_gpio_write(GREEN_LED,LOW);
       }

}
```

II.3.2 QueryContext.

La función *queryContext* desempeña una función distinta dependiendo del dispositivo. En los siguientes puntos utilizarán unos formularios XML concretos, sin embargo, estos se podrán cambiar de forma que sea posible interactuar con la entidad y atributos deseados.

II.3.2.1 Arduino.

En el caso de Arduino se encarga de realizar la conexión con el servidor, enviar por HTTP el formulario correspondiente a una petición *queryContext,* analiza la respuesta del

servidor y busca en ella el valor de contexto deseado. Por lo tanto es una función que devuelve un valor numérico.

En primer lugar realiza la conexión mediante *client.connect(server,1026)* y a la vez comprueba si ésta es posible.

```
int queryContext(){
  int val=0;
  if (client.connect(server, 1026)) {
```

A continuación se envía el formulario XML mediante *client.print()*.

```
    Serial.println("Query context");
    client.println("POST /NGSI10/queryContext HTTP/1.1");
    client.println("Host: 130.206.83.2:1026");
    client.println("User-Agent: Arduino/1.1");
    client.println("Connection: close");
    client.println("Content-Type: application/xml");
    client.print("Content-Length: ");
    client.println("274");
    client.println();
    client.println("<?xml version=\"1.0\" encoding=\"UFT-8\"?>");
    client.println("<queryContextRequest>");
    client.println("<entityIdList>");
    client.println("<entityId type=\"UPCT:PARKING\" isPattern=\"false\">");
    client.println("<id>UPCT:PARKING:ED</id>");
    client.println("</entityId>");
    client.println("</entityIdList>");
    client.println("<attributeList>");
    client.println("<attribute>car_out</attribute>");
    client.println("</attributeList>");
    //client.println("<attributeList/>");
    client.println("</queryContextRequest>");
    client.println();
    Serial.println("Done");
```

Ahora se espera hasta recibir la respuesta del servidor, y en caso de no ocurrir se imprime un mensaje de error.

```
    while(!client.available());
  }
  else{
    Serial.println("ERROR");
  }
```

Se analiza la respuesta conforme se va recibiendo del servidor. Para ello, se van almacenando los caracteres en el string *line*. Cuando se registra que un carácter es 10 (salto de línea en ASCII) se analiza la línea en busca de la etiqueta <*contextValue*>, mediante otra función llamada *searchValue(line*, val), y si se encuentra el valor se almacena en la variable *val*. Finalmente, la función devuelve dicho valor numérico y cierra la conexión.

```
  while (client.available()) {
    char c = client.read();
    line=line + c;
    if(c==10){
      val=searchValue(line,val);
      line="";
    }
  }
    Serial.println("Finished...");
    client.stop();
```

```
  return val;
}
```

La función *searchValue()* se encarga de buscar en la línea, pasada como argumento (el carácter *<*). A continuación, comprueba si lo que sigue a dicho carácter es el texto *contextValue*. En caso afirmativo, busca la etiqueta *</*, que es donde acaba el valor de contexto, y guarda en la variable *val* el contenido entre > y </.

```
int searchValue(String s,int i) {
  int beginning,ending;
  String val;
  beginning=s.indexOf('>');
  ending=s.indexOf('<', beginning+1);
  if(s.startsWith("contextValue",s.indexOf('<')+1)){
    val =s.substring(beginning+1,ending);
    return val.toInt();
  }
  else{
    return i;
  }
}
```

II.3.2.2 Raspberry Pi.

La función *queryContext()* tiene una implicación distinta en Raspberry Pi. En este caso lo que hace es devolver el valor del formulario XML correspondiente en forma de variable tipo string. La utilidad de dicha función es guardar ese formulario en otra variable, que es enviada posteriormente como se muestra en el Capítulo 4.

```
std::string queryContext(){

        return "<?xml version=\"1.0\" encoding=\"UTF-8\"?>\n\
       <queryContextRequest>\n\
       <entityIdList>\n\
       <entityId type=\"UPCT:PARKING\" isPattern=\"false\">\n\
       <id>UPCT:PARKING:D2</id>\n\
       </entityId>\n\
       </entityIdList>\n\
       <attributeList>\n\
       <attribute>reserved</attribute>\n\
       </attributeList>\n\
       </queryContextRequest>\n";
}
```

II.3.3 UpdateContext.

UpdateContext() es una función muy parecida a la anterior, que en este caso recibe argumentos numéricos medidos en los dispositivos para que sean enviados a FI-WARE. Nuevamente, es distinta la implementación en Arduino y Raspberry Pi.

II.3.3.1 Arduino.

La función *updateContext()* en Arduino es una función que se encarga de conectar con el servidor, enviar el formulario XML y, si se desea, observar la respuesta del servidor. Tiene como argumentos los valores de contexto a actualizar que son integrados en el formulario.

```
        void updateContext(int val){
          if (client.connect(server, 1026)) {
            Serial.println("Update context");
            client.println("POST /NGSI10/updateContext HTTP/1.1");
            client.println("Host: 130.206.83.2:1026");
            client.println("User-Agent: Arduino/1.1");
            client.println("Connection: close");
            client.println("Content-Type: application/xml");
            client.print("Content-Length: ");
            client.println("460");
            client.println();
            client.println("<?xml version=\"1.0\" encoding=\"UFT-8\"?>");
            client.println("<updateContextRequest>");
            client.println("<contextElementList>");
            client.println("<contextElement>");
            client.println("<entityId type=\"UPCT:PARKING\" isPattern=\"false\">");
            client.println("<id>UPCT:PARKING:ED</id>");
            client.println("</entityId>");
            client.println("<contextAttributeList>");
            client.println("<contextAttribute>");
            client.println("<name>car_in</name>");
            client.println("<type>lot</type>");
            client.print("<contextValue>");
            client.print(val);
            client.println("</contextValue>");
            client.println("</contextAttribute>");
            client.println("</contextAttributeList>");
            client.println("</contextElement>");
            client.println("</contextElementList>");
            client.println("<updateAction>UPDATE</updateAction>");
            client.println("</updateContextRequest>");
            client.println();

            Serial.println("Done");
            while(!client.available
          }
          else{
            Serial.println("ERROR");
          }

          //Use when printing Http response is required********
          //while (client.available()) {
          //char c = client.read();
          //Serial.write(c);
          //}

            Serial.println("Finished...");
            client.stop();
        }
```

II.3.3.2 Raspberry Pi.

En el caso de Rasbperry Pi, al igual que la función *queryContext()*, en este caso la función se encarga de devolver un string con el formulario XML, que posteriormente se enviará con otra función ya comentada con anterioridad.

Para la inserción de los valores que se pasan como argumentos dentro del formulario se utilizan variables de la librería *stringstream*.

```
std::string updateContext(float val1,float val2,int val3,int val4){
  std::stringstream ss;
  std::string XML;

  ss << "<?xml version=\"1.0\" encoding=\"UTF-8\"?>\n\
  <updateContextRequest>\n\
  <contextElementList>\n\
  <contextElement>\n\
  <entityId type=\"UPCT:PARKING\" isPattern=\"false\">\n\
  <id>UPCT:PARKING:D2</id>\n\
  </entityId>\n\
  <contextAttributeList>\n\
  <contextAttribute>\n\
  <name>temperature</name>\n\
  <type>centigrade</type>\n\
  <contextValue>" << val1;
  XML=ss.str();
  ss << "</contextValue>\n\
  </contextAttribute>\n\
  <contextAttribute>\n\
  <name>humidity</name>\n\
  <type>relative</type>\n\
  <contextValue>" << val2;
  XML=XML+ss.str();
  ss <<"</contextValue>\n\
  </contextAttribute>\n\
  <contextAttribute>\n\
  <name>occupied</name>\n\
  <type>value</type>\n\
  <contextValue>"<< val3;
  XML=XML+ss.str();
  ss <<"</contextValue>\n\
  </contextAttribute>\n\
  <contextAttribute>\n\
  <name>reserved</name>\n\
  <type>value</type>\n\
  <contextValue>" << val4;
  XML=ss.str();
  XML=XML+"</contextValue>\n\
  </contextAttribute>\n\
  </contextAttributeList>\n\
  </contextElement>\n\
  </contextElementList>\n\
  <updateAction>UPDATE</updateAction>\n\
  </updateContextRequest>\n";

  return XML;
}
```

Al igual que en los casos anteriores, para enviar datos a otra entidad con otros atributos simplemente se deben modificar los argumentos que se pasan a la función y modificar los valores de entidad y tipo del XML guardado en la variable *XML*.

II.3.4. GetTemp y getHumid.

Son dos funciones empleadas en el dispositivo 4, basado en Rasbperry Pi, para leer la temperatura y humedad relativa. Están basadas en lo expuesto en el apartado II.2.5 de este mismo anexo.

La función *getTemp()* es de la forma:

```
float getTemp(void){
       unsigned char noError = 1;
       value humi_val,temp_val;
       delay(20);
       SHT1x_InitPins();
```

```
        SHT1x_Reset();

        temp_val.f = (float)temp_val.i;
        humi_val.f = (float)humi_val.i;

        SHT1x_Calc(&humi_val.f, &temp_val.f);

        printf("Temperature: %0.2f%cC\n",temp_val.f,0x00B0);

        return temp_val.f;
}
```

La función *getHumid():*

```
float getHumid(void){
        unsigned char noError = 1;
        value humi_val,temp_val;
        delay(20);
        SHT1x_InitPins();
        SHT1x_Reset();

        temp_val.f = (float)temp_val.i;
        humi_val.f = (float)humi_val.i;

        SHT1x_Calc(&humi_val.f, &temp_val.f);

        printf("Humedad relativa: %0.2f % \n",humi_val.f);

        return humi_val.f;
}
```

II.3.5. GetDistance.

Se trata de una función que devuelve una lectura de distancia. Se utiliza en los dispositivos basados en Arduino. Esta función es una adaptación de lo expuesto en el apartado II.2.3 de este anexo.

```
int getDistance(){
  int dist=0;
  Wire.beginTransmission(112);

  Wire.write(byte(0x00));
  Wire.write(byte(0x51));
  Wire.endTransmission();

  delay(70);

  Wire.beginTransmission(112);
  Wire.write(byte(0x02));
  Wire.endTransmission();

  Wire.requestFrom(112, 2);

  if(2 <= Wire.available()){
    dist = Wire.read();
    dist= dist << 8;
    dist |= Wire.read();
  }
  return dist;
}
```

II.4 Desarrollo del widget UPCT Parking Interface.

En el Capítulo 4 se muestra la interfaz y el funcionamiento del widget. En este anexo se expone la estructura de éste (ver Ilustración II.1) y se va a describir el contenido de los archivos que forman parte de ella. Es necesario emplear HTML, CSS, Javascript y jQuery para el desarrollo del widget [14].

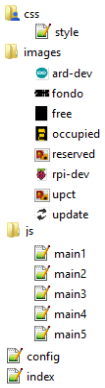

Ilustración I.1. Vista de la estructura del widget

II.4.1 Archivo de interfaz HTML y hoja de estilos CSS.

Para la creación de la interfaz del widget se emplea el lenguaje de maquetación web HTML, y si se estima necesario, hojas de estilos CSS.

En la aplicación creada para el caso práctico del presente proyecto se utilizan dimensiones fijas medidas en píxeles, debido a la ventaja que supone para alinear elementos.

El documento HTML, llamado *index.html,* comienza con la importación de los scripts de javascript, que más tarde se expondrán, y se vincula con la hoja de estilos CSS ubicada en *css/style.css*

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
    <head>
        <meta http-equiv="Content-Type" content="application/xhtml+xml; charset=UTF-
8"></meta>
        <title>Parking UPCT</title>
       <script
src="https://ajax.googleapis.com/ajax/libs/jquery/1.10.2/jquery.min.js"></script>
        <script type="text/javascript" src="js/main1.js"></script>
              <script type="text/javascript" src="js/main2.js"></script>
              <script type="text/javascript" src="js/main3.js"></script>
              <script type="text/javascript" src="js/main4.js"></script>
              <script type="text/javascript" src="js/main5.js"></script>

        <link rel="stylesheet" type="text/css" href="css/style.css" />
    </head>
```

En el cuerpo del documento HTML, se crean elementos capa, *div,* que contendrán los diferentes elementos. El primero de ellos cuyo id es *container,* contiene al resto de capas y tiene dimensiones fijas de 735x325, igual al tamaño del fondo de la web. Además, el tamaño de la web tiene esas mismas dimensiones. Dentro de él se coloca el botón de actualización (Botón 3 mostrado en el Capítulo 4). A su vez, dentro de *container* se crean otras subcapas que se adaptan a cada una de las plazas de garaje representadas en el fondo y cuyos id son *dev1* y *dev*. Cada subcapa tiene un botón que corresponderán con los botones 1 y 2 en el Capítulo 4.

```
        <body>

        <div id="container" >
        <form>
               <input type="button" class="button" id="update">
        </form>
        <div id="lot1" >
        <form>
               <input type="button" class="button-dev" id="dev1">
        </form>

        </div>
        <div id="lot2" >
        <form>
               <input type="button" class="button-dev" id="dev2">
        </form>
        </div>
```

Finalmente, una última capa se ubica en el centro de *container* y se utiliza para mostrar información al usuario. Ésta contiene varias etiquetas párrafo y título con identificadores propios para poder acceder a ellos con jQuery y cambiar su contenido.

```
        <div id="cartel" >
        <h1 id="title"></h1>
        <p id="content1"> </p>
        <p id="content2"> </p>
        <p id="content3"> </p>
        </div>
        </div>
    </body>
</html>
```

La hoja de estilos CSS aporta características a las capas mencionadas, para que se alineen de forma adecuada, añade el fondo al widget, edita el comportamiento de los botones y sus imágenes y los alinea.

```
        body {width:735px; height:373 px; background-image:url('../images/fondo.png');
        background-repeat: no-repeat;font-family: Arial;}

        p {font: 70% Arial; color:#FFFFFF;}
        h1 { font: bold 100% Arial; color:#FFFFFF;}

        h2 { font: bold 115% Arial; color:#6495ed;}

        #update{float:right; width: 45px; height: 45px; margin-right:-15px;
        margin-top:-20px; background-image:url('../images/update.png');
        background-position: center center; background-color: #FFFF00;
        box-shadow: 3px 3px 3px #888888; cursor:pointer; cursor:hand;}

        #container{ width:697px; height:329px; position:relative; left: 13px; top: 14px;}

        #lot1{ position:relative; left:38px; width: 84px; height: 110px;
```

```
        background-image:url('../images/free.png'); background-position: center center;}

        .button-dev{ width: 27px; height: 27px; cursor:pointer; cursor:hand;
        border-radius: 7px;}

        #dev1{background-image: url('../images/ard-dev.png');
        background-position: center center; background-repeat: no-repeat;
        position:relative; left: 31px;}

        #dev2{background-image: url('../images/rpi-dev.png');
        background-position: center center; background-repeat: no-repeat;
        position:relative; left: 27px;}

        #lot2{position:relative; left:219px; top:-110px; width: 80px; height: 110px;
        background-image:url('../images/free.png'); background-position: center center;}

        #cartel{width: 300px; height: 80px; position:absolute; right: 200px;top:125px;}
```

II.4.2 Archivo configuración XML.

Las aplicaciones de *Wirecloud* necesitan un archivo de configuración en formato XML que aporte información sobre el widget. En este caso, la aplicación no tiene entradas ni salidas para poder enlazarla con otras. Por lo tanto, los datos aportados por el archivo son del desarrollador, iconos y el tamaño del widget una vez importado al espacio de trabajo de FI-Lab.

```
<?xml version="1.0" encoding="UTF-8"?>

<!-- The Template core element. This element is mandatory -->
<Template xmlns="http://wirecloud.conwet.fi.upm.es/ns/template#">

    <!-- Catalog.ResourceDescription element. This element defines the core metadata of
the widget -->
    <Catalog.ResourceDescription>
        <!-- Vendor: Company that distributes the widget -->
        <Vendor>UPCT</Vendor>
        <!-- Name: Name of the widget -->
        <Name>UPCT Parking Interface rev 2</Name>
        <!-- Version: Current widget version number-->
        <Version>4.4</Version>
        <!-- DisplayName: Name shown in the UI of the widget -->
        <DisplayName>UPCT Parking Interface rev 2</DisplayName>
        <!-- Author: Widget developer(s) -->
        <Author>Enrique Hernández</Author>
        <!-- Mail: Developer's email address -->
        <Mail>quiquehz@gmail.com</Mail>
        <!-- Description: Full widget description to be shown in the catalogue -->
        <Description>Monitoring parking status rev 2</Description>
        <!-- Absolute or template-relative path to the image shown in the catalogue -->
        <ImageURI>images/upct.png</ImageURI>
        <!-- Absolute or template-relative path to the image to be used in smartphones -
->
        <iPhoneImageURI>images/upct.png</iPhoneImageURI>
        <!-- Absolute or template-relative path to the widget's documentation -->
        <WikiURI>doc/index.html</WikiURI>

       <Requirements>
            <Feature name="NGSI"/>
        </Requirements>

    </Catalog.ResourceDescription>

    <!-- Platform.Preferences element. It defines user preferences -->
    <Platform.Preferences>
    </Platform.Preferences>

    <!-- Platform.StateProperties element. It defines user preferences -->
    <Platform.StateProperties>
```

```
    </Platform.StateProperties>

    <!-- Platform.Wiring element. It defines both the widget inputs and outputs -->
    <Platform.Wiring>
    </Platform.Wiring>

    <!-- Platform.Link element. It binds the template with the actual widget's source
code. -->
    <Platform.Link>
        <!-- XHTML element. It has the href attribute that links to the source code of
the widget. -->
        <!--                contenttype and cacheable attributes are optional -->
        <XHTML href="index.html"/>

    </Platform.Link>

    <!-- Platform.Rendering element. It specifies the default width and height of the
widget -->
    <Platform.Rendering width="10" height="30"/>

</Template>
```

II.4.3 Archivos Javascript y jQuery.

Aquí es donde se encuentra la parte más importante del widget. Estos archivos son los que se encargan de recibir la información de contexto de las entidades y plasmarla en la interfaz modificando características del documento HTML antes creado.

Son 5 los archivos empleados en este widget, *main1.js, main2.js, main3,js, main4.js* y *main5.js*. Cada uno de ellos contiene una función creada con ayuda de la API NGSI de FI-WARE.

- Main1.js. Se encarga de mostrar las últimas condiciones de temperatura y humedad relativa registradas en la plaza donde se encuentra el dispositivo Arduino al hacer click sobre el botón de esa misma plaza (Botón 1). Para ello, consulta la entidad UPCT:PARKING:D1.

- Main2.js. Muestra las últimas condiciones de temperatura y humedad relativa registradas en la plaza donde se encuentra el dispositivo Raspberry Pi al hacer click sobre el botón de esa misma plaza (Botón 2). Para ello, consulta la entidad UPCT:PARKING:D2.

- Main3.js. Su función es mostrar en el div central el número de plazas libres, como resultado de consultar la entidad UPCT:PARKINGED al pulsar sobre el botón *update* (UPCT:PARKING:ED).

- Main4.js. Al pulsar sobre el botón *update* se cambia el fondo (background) de la capa *dev1* en función de si la plaza está ocupada o no. De esta forma el usuario observará una imagen distinta en cada plaza si está ésta ocupada, libre o reservada.

- Main5.js. Realiza la misma función que Main4.js, pero en este caso afecta a la capa *dev2*.

En todos los archivos la estructura del código es el mismo. Implementan una función que se ejecuta al hacer click en un determinado botón utilizando la sintaxis jQuery:

```
$("#ID_BOTON").click(function(){getEntityInfo(connection);}
```

De forma que cambiando *ID_BOTON* por el identificador del botón requerido, se ejecute lo que hay entre corchetes. En todos los casos, eso que se ejecuta entre corchetes es a su vez la función proveniente de la API mencionada, *getEntityInfo()*. Ésta es la que se encarga de crear la conexión con el servidor y recibir la información de contexto. Su sintaxis es

```
var getEntityInfo = function getEntityInfo(connection) {

                        var entityList = [{type: 'UPCT:PARKING', id: 'UPCT:PARKING:ED'},]
                        connection.query(entityList, null, {onSuccess: function(response){
                }
```

Cuando se realiza correctamente, se recibe una respuesta del servidor en forma de array JSON (array de un solo elemento) y se almacena en la variable *response*. De esta forma, consultado esta variable se pueden obtener los valores de contexto.

Así, el contenido de *main1.js* se crea una *query* a la entidad *UPCT:PARING:D1* y se buscan sus valores *temperature* y *humidity* mostrándolos en los párrafos *content1 y content2*.

```
(function () {
        var getEntityInfo = function getEntityInfo(connection) {
                var entityList = [{type: 'UPCT:PARKING', id: 'UPCT:PARKING:D1'},]
                connection.query(entityList, null, {
                        onSuccess: function(response){
                                $("#title").text("Device 1 (Arduino)");

                                $("#content1").text("Temperature: "+
response[0].attributes[0].contextValue+ " ºC");

                                $("#content2").text("Relative humidity: " +
response[0].attributes[1].contextValue+ " %");
                        }});}
        $(document).ready(function(){
                var connection = new NGSI.Connection('http://130.206.83.2:1026');
                $("#dev1").click(function(){getEntityInfo(connection);});
        });
})();
```

Es importante mencionar, que al tratarse de un array, para buscar un valor de contexto en la respuesta, se debe buscar en el primer elemento de dicho array. Es por ello que se utiliza *response[0].attributes[0]…,* en lugar de *response.attributes[0]…,* como ocurría si fuera un objeto JSON.

Exactamente igual ocurre con el dispositivo Raspberry Pi en *main2.js*

```
(function () {

        var getEntityInfo = function getEntityInfo(connection) {
```

```
                        var entityList = [{type: 'UPCT:PARKING', id: 'UPCT:PARKING:D2'},]
                        connection.query(entityList, null, {onSuccess: function(response){

                                $("#title").text("Device 2 (Raspberry Pi)");

                                $("#content1").text("Temperature: "+
response[0].attributes[0].contextValue+ " ºC");

                                $("#content2").text("Relative humidity: " +
response[0].attributes[1].contextValue+ " %");

                }});
        }

        $(document).ready(function(){
                var connection = new NGSI.Connection('http://130.206.83.2:1026');
                $("#dev2").click(function(){getEntityInfo(connection);});
        });
})();
```

En el tercer archivo, *main3.js,* el botón que activa la función es *update*. Además en este caso, se realiza una operación matemática para obtener las plazas restantes. A 14, que es el número total de plazas se le restan las ocupadas (*car_in – car_out)*. Para ello, se deben convertir los objetos JSON *car_in* y *car_out* a valor numérico con *Number()*. Finalmente, se vacía el contenido de los párrafos *content1* y *content2* para evitar interferencias con los archivos anteriores.

```
(function () {

        var getEntityInfo = function getEntityInfo(connection) {

                        var entityList = [{type: 'UPCT:PARKING', id: 'UPCT:PARKING:ED'},]
                        connection.query(entityList, null, {onSuccess: function(response){
                        var max_lots = 14;
                        var car_out= response[0].attributes[0].contextValue;
                        var car_in= response[0].attributes[1].contextValue;
                        var lots= 14-Number(car_in)+Number(car_out);
                                $("#title").text("PARKING AVAILABILITY: "+ lots +" LOTS");
                                $("#content1").text("");
                                $("#content2").text("");
                }});
        }

        $(document).ready(function(){
                var connection = new NGSI.Connection('http://130.206.83.2:1026');
                $("#update").click(function(){getEntityInfo(connection);});
        });
})();
```

En *main4.js* se obtiene el valor de *occupied* de la entidad *UPCT:PARKING:D1*. Dependiendo de su valor, se cambiará la imagen de fondo de la capa.

```
(function () {

        var getEntityInfo = function getEntityInfo(connection) {

                        var entityList = [{type: 'UPCT:PARKING', id: 'UPCT:PARKING:D1'},]
                        connection.query(entityList, null, {onSuccess: function(response){
                        var occupied= response[0].attributes[2].contextValue;
                        if(occupied>0){
                        $('#lot1').css('background-image','url(images/occupied.png)');
                        }else{
                        $('#lot1').css('background-image','url(images/free.png)');
                        }
                }});
        }
```

```
        $(document).ready(function(){
                var connection = new NGSI.Connection('http://130.206.83.2:1026');
                $("#update").click(function(){getEntityInfo(connection);});
        });
})();
```

Para el último archivo, *main5.js,* ocurre algo muy similar que para *main4.js*. Sin embargo, en este caso hay que añadir la posibilidad de que la plaza esté reservada. Ello se comprueba mediante la consulta del atributo *reserved*.

```
(function () {

        var getEntityInfo = function getEntityInfo(connection) {

                        var entityList = [{type: 'UPCT:PARKING', id: 'UPCT:PARKING:D2'},]
                        connection.query(entityList, null, {onSuccess: function(response){
                        var occupied= response[0].attributes[2].contextValue;
                        var reserved= response[0].attributes[3].contextValue;
                        if(occupied>0){
                        $('#lot2').css('background-image','url(images/occupied.png)');
                        }else{
                        $('#lot2').css('background-image','url(images/free.png)');
                        if(reserved>0){
                        $('#lot2').css('background-image','url(images/reserved.png)');
                        }
                        }
                }});
        }

        $(document).ready(function(){
                var connection = new NGSI.Connection('http://130.206.83.2:1026');
                $("#update").click(function(){getEntityInfo(connection);});
        });
})();
```

Bibliografía

[1] ARDUINO.CC. HARDWARE. Web de Arduino. *Documentación necesaria para la descripción del hardware.* [consulta: 12 de Agsoto de 2014]. Disponible en: http://arduino.cc/en/Main/Products?from=Main.Hardware

[2] PINGUINO.CC. HARDWARE. Web de Pinguino. *Documentación para la descripción del hardware.* [consulta: 12 de Agsoto de 2014]. Disponible en: http://www.pinguino.cc/

[3] RASPBERRYPI.ORG. HARDWARE. Web de Raspbery Pi. *Documentación necesaria para la descripción del hardware.* [consulta: 14 de Agosto de 2014]. Disponible en: http:// www.raspberrypi.org/tag/hardware/

[4] BEAGLEBOARD.ORG. HARDWARE. Web de Beagle Board. *Documentación necesaria para la descripción del hardware.* [consulta: 15 de Agosto de 2014]. Disponible en: http://beagleboard.org/bone

[5] LIBELIUM.COM. HARDWARE. Web de Waspmote. *Documentación necesaria para la descripción del hardware.* [consulta: 15 de Agosto de 2014]. Disponible en: http://www.libelium.com/es/products/waspmote/

[6] WIKIPEDIAORG. *Documentación para la descripción de distintas plataformas.* [consulta: 17 de Agosto de 2014]. Disponible en: http://www.es.wikipedia.org/

[7] FI-WARE.ORG. *Información general sobre FI-WARE.*[consulta: 18 de Agosto de 2014]. Disponible en: https://www.fi-ware.org

[8] Wiki de FI-WARE. FI-WARE Architecture. *Documentación sobre la arquitectura general de FI-WARE.* [consulta: 09 de Agosto de 2014]. Disponible en: https://forge.fi-ware.org/plugins/mediawiki/wiki/fiware/index.php/

[9] CATALOGO FI-WARE. *Información sobre los GE de FI-WARE*. [consulta: 18 de Agosto de 2014]. Disponible en: https://catalogue.fiware.org

[10] BRUCE ECKEL, *Thinking in C++, Vol 1*. Ed. Mindview, Inc. Año 2000. [consulta: 17 de Agosto de 2014].

[11] PUBLISH/SUBSCRIBE CB. USERS AND PROGRAMMERS GUIDE. FI-WARE WIKI. *Guía para crear, actualizar, suscribir y consultar entidades*. [consultado: 15 Octubre de 2013]. Disponible en: https://forge.fi-ware.org/plugins/mediawiki/wiki/fiware/index.php/Publish/Subscribe_Broker_-_Orion_Context_Broker_-_User_and_Programmers_Guide

[12] ARDUINO.CC. SOFTWARE. Foros de Arduino. *Librerías necesarias para la conexión de sensores*. [consulta: 15 de Julio de 2014] Disponible en: http://forum.arduino.cc/

[13] JOHM GEEK NZ. *Librería para la conexión del sensor SHT11 a Raspberry Pi*. [consulta: 03 de Septiembre de 2014]. Disponible en: https://www.john.geek.nz/2012/08/reading-data-from-a-sensirion-sht1x-with-a-raspberry-pi/

[14] JON DUCKETT. *Javascript and jQuery. Interactive Front-End Web Development.* Ed. Copyrighted material. Año 2014. [consulta: 10 de Septiembre de 2014].

Páginas de ayuda.

STACKOVERFLOW. *Web de preguntas y respuestas de programación*. [consultado: a lo largo de todo el proyecto]. Disponible en: http://www.stackoverflow.com

PORTAL E-LEARNING DE FI-WARE. Cursos e información sobre FI-WARE. . [consultado: a lo largo de todo el proyecto]. Disponible en: http://edu.fi-ware.org/