Universitat Oberta de Catalunya

uoc.edu

Sistema monitor para personas dependientes

César López Bermúdez
M.U. Ingeniería de telecomunicaciones
Área de electrónica

Aleix López Antón

Carlos Monzo Sánchez

10 de Enero de 2019

FICHA DEL TRABAJO FINAL

Título del trabajo:	*Sistema monitor de personas dependientes*
Nombre del autor:	*César López Bermúdez*
Nombre del consultor/a:	*Aleix López Antón*
Nombre del PRA:	*Carlos Monzo Sánchez*
Fecha de entrega (mm/aaaa):	01/2019
Titulación::	*M.U. Ingeniería de telecomunicaciones*
Área del Trabajo Final:	*Electrónica*
Idioma del trabajo:	***Castellano***
Palabras clave	*wereable, IoT, embebido*

Resumen del Trabajo :

A lo largo de los últimos años la preocupación por parte de entes públicos en cuanto a la población dependiente y de tercera edad residente en el rural gallego no ha dejado de crecer. En una distribución de la población rural extremadamente atomizada en pequeños pueblos satélites de los ayuntamientos centrales, con grandes distancias entre ellos y sobre un terreno montañoso con condiciones climatológicas adversas tanto en invierno(bajas temperaturas, lluvia, helada) como en verano(altas temperaturas) sitúan a la población rural de mayor edad en situación de riesgo.

Se pretende por tanto de desarrollar un prototipo de sistema embebido o IoT bien conocido comúnmente como *wereable* tratándose en este caso de una pulsera. Tratando de minimizar su coste económico, enlazado con un nodo central, capaz de recabar y almacenar datos de monitoreo de personas pertenecientes al público objetivo del proyecto mediante la adquisición y tratamiento de los datos muestreados a través de los diferentes biosensores que se determinarán tales como, acelerómetro, giroscopio y magnetómetro(“la persona está inmóvil?”, ”Se ha desvanecido?”, etc...) y otros que se analizarán(viabilidad, coste, etc...) que permitan obtener datos del entorno del usuario como por ejemplo la temperatura a la que éste se encuentra expuesto.

Se pretende que el sistema conste de un módulo GPS que permita obtener las coordenadas del usuario en caso emergencia para así poder proceder a su rescate. También constará de un módulo de comunicaciones a determinar (GSM, 3G…) para enviar los datos obtenidos y la generación de alarmas al nodo central recopilador de datos.

Se analizarán diversas tecnologías para el desarrollo del prototipo, obteniéndose un prototipo a modo de prueba de concepto sobre tecnologías aptas para el desarrollo posterior de un proyecto comercial.

Abstract:

Over the past few years, the concern of public bodies regarding the dependent and elderly population living in rural Galicia has not stopped growing. In a distribution of the extremely atomized rural population in small satellite towns of the central town halls, with great distances between them and on a mountainous terrain with adverse weather conditions both in winter (low temperatures, rain, frost) and in summer (high temperatures) They place the older rural population at risk.

It is therefore intended to develop a prototype embedded system or IoT well known commonly as wereable in this case being a bracelet. Trying to minimize its economic cost, linked to a central node, capable of collecting and storing monitoring data of people belonging to the target audience of the project through the acquisition and processing of the sampled data through the different biosensors that will be determined such as, accelerometer, gyroscope and magnetometer ("the person is immobile?", "Has vanished?", etc ...) and others that will be analyzed (viability, cost, etc ...) that allow to obtain data from the user's environment such as for example, the temperature at which it is exposed.

It is intended that the system consists of a GPS module that allows obtaining the user's coordinates in case of an emergency so that they can proceed to their rescue. It will also consist of a communications module to be determined (GSM,

3G ...) to send the obtained data and the generation of alarms to the central data collector node.

Different technologies will be analyzed for the development of the prototype, obtaining a prototype as a proof of concept on technologies suitable for the subsequent development of a commercial project.

Índice

Lista de figuras

Lista de Tablas

1. Introducción

1.1 Contexto y justificación del Trabajo

El proyecto propuesto pretende dar continuación a un proyecto anterior [CES] que constituía una prueba de concepto limitada, para una solución ante un problema social detectado en el ámbito rural de la comunidad de Galicia.

A lo largo de la última década, la población gallega residente en ámbito rural viene sufriendo un envejecimiento cada vez mayor. El éxodo de población joven hacia áreas más urbanas en busca de mejores servicios y situación laboral, concluye en una población fijada en el rural, de elevada edad y sometida a condiciones de aislamiento, debido fundamentalmente a la distribución atomizada de ayuntamientos en zonas de montaña.

En la siguiente ilustración obtenida de [CES] se observa la proporcionalidad inversa entre la densidad de población, rural frente a urbano, y el porcentaje de individuos mayores de 65 años por cada cien menores de 20.

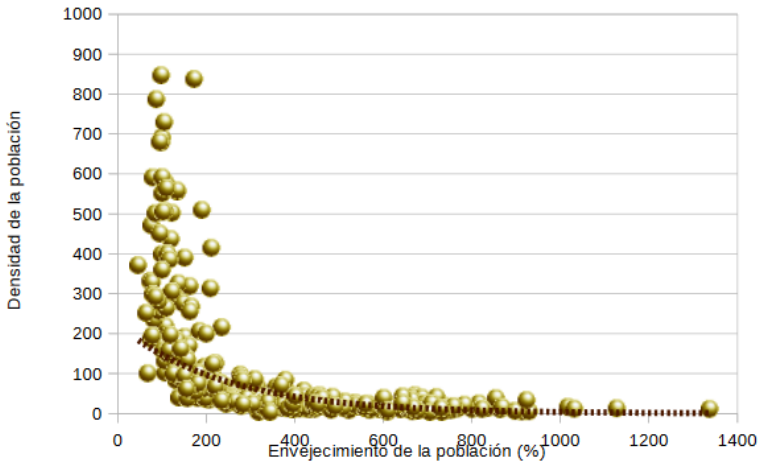


Ilustración 1: Edad frente a densidad de población

A esta evolución demográfica de la población rural, se le suma la débil provisión de servicios elementales en dicho ámbito, tales como la asistencia sanitaria, cada vez más precaria, llegando a requerirse tiempos de viaje por carretera superiores a una hora para acceder a un punto de atención continuada con asistencia médica básica.

A esto ha de añadirse la deficiente situación de infraestructuras como ocurre con el caso de las vías de comunicación por carretera, que en ocasiones no se encuentran asfaltadas.

Todo esto genera una población de individuos de avanzada edad, residiendo en condiciones de aislamiento, que en gran medida viven en soledad.

En los últimos años vienen observándose situaciones de extravío de estos ciudadanos, en ocasiones por desorientación, que llegan incluso a concluir con fatal desenlace. Esto ha sido debidamente recogido por medios de prensa, tal y como se citan en [CES], debido a la alarma social generada.

La aplicación de nuevas tecnologías como la denominada Internet of Things (IoT), pueden constituir una buena herramienta para combatir tal situación de aislamiento y corregir las cifras de extravíos y fallecimiento de ciudadanos errantes.

1.2 Descripción del sistema

Tal y como se ha mencionado en el punto anterior, el proyecto propuesto consiste en una continuación de un proyecto anterior [CES].

Si bien en este caso se pretende mejorar el sistema utilizando tecnologías disponibles para el desarrollo de sistemas embebidos que permitan reducir el tiempo de desarrollo, y por tanto permitan acortar el tiempo time-to-market.

Además, se pretende emplear tecnologías de nube o cloud para el almacenamiento de datos y la gestión de múltiples dispositivos sensores.

La arquitectura del sistema completo consiste en un conjunto de dispositivos de tipo vestible o *wereable*, que portarán los usuarios. Estos consistirán dispositivos de monitorización de personas (DMP) a modo de pulsera provistos de sensores, capaces de recolectar datos que permitan modelar el comportamiento humano.

Dichos datos serán transmitidos de forma inalámbrica a un nodo central general (NCG) de almacenamiento vía GSM.

Además se pretende implementar posicionamiento GPS para disponer de geoposicionamiento en caso de requerirse la localización del usuario.

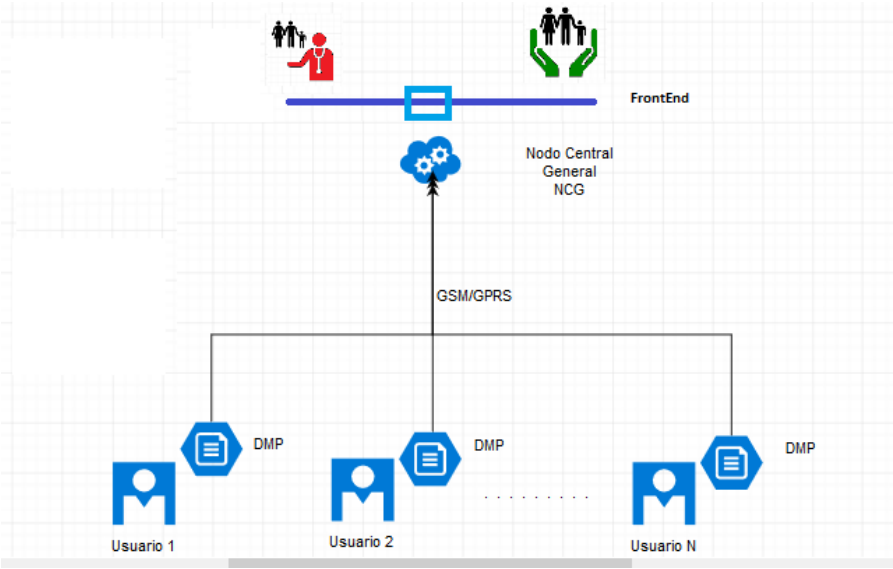


Ilustración 2: Arquitectura de sistema

La comunicación entre DMPs y el NCG será bidireccional, de este modo, los DMPs enviarán los datos recolectados de los sensores al servicio cloud, y la aplicación del NCG a su vez, será capaz de enviar parámetros de configuración y otros comandos al cada DMP.

Por otro lado, el sistema completo contaría con un frontend desarrollado sobre el NCG con vistas cliente para servicios médicos y otros de tipo asistencial. De este modo, cualquier médico que trate al usuario como paciente podría ordenar toma de medicación o concertar cita médica, y

el propio NCG sería encargado de transmitir dicha información al DMP correspondiente a dicho usuario. Por su parte, servicios asistenciales, tanto públicos como privados, podrían acceder a información sobre el paciente e interactuar con el a través del intercambio de mensajes entre NCG y DMP.

1.3 Descripción del alcance del trabajo

Este proyecto pretende consistir en una prueba de concepto limitada sobre el sistema completo planteado en el punto anterior. En estos términos los tres niveles de la arquitectura mostrada en la ilustración 2 se desarrollarán con las limitaciones y restricciones siguientes:

- DMPs: Los dispositivos terminales o wereables de usuario, recabarán datos de sensorización siguientes: acelerómetro, giroscopio, magnetómetro, presión, temperatura, humedad, nivel de luz ambiente, nivel de batería, situación de carga de batería. Además de posición GPS si se requiere.

  Tras la primera conexión con el NCG, éste le transmitirá información de configuración al DMP particularizada para el mismo. El DMP comenzará a transmitir al NCG ,vía GSM/GPRS, la información que recaba enformato JSON.

  El DMP mostrará al usuario mediante pantalla, el estado de la conexión GSM, conexión con NCG, y estado de la batería. Además en caso de exposición a temperaturas extremas tanto superiores como inferiores prefijadas, emitirá mediante pantalla un aviso de emergencia al usuario para que éste se regrese a una situación de seguridad.

- El NCG se implementará mediante el desarrollo de una pequeña aplicación capaz de acceder a los datos publicados por el conjunto de todos los DMP. Cuando un nuevo DMP se conecte al servicio en la nube, el NCG le enviará información de configuración. No se implementa el procesado de los datos

obtenidos ya que esto, debido a la magnitud del trabajo requerido, exige un proyecto en sí a fin de implementar técnicas de análisis de datos masivos como puede ser la aplicación de redes neuronales.

- Frontend. No se implementa. La aplicación que simula el NCG, (NCG_SIM), realizará la simulación de algunas de las posibilidades de mensajes a transmitir al DMP. (toma de medicación, cita con el médico, etc…)

1.4 Objetivos del trabajo

Se pretende lograr los siguientes objetivos principales:

- Lograr un prototipo válido para testing que cumpla los requisitos establecidos.
- Realizar análisis del estado del arte e iniciativas similares al proyecto que pudieran existir.
- Identificar y comparar diferentes plataformas hardware para desarrollo de dispositivos IoT que permitan reducir el tiempo desde diseño a testing.
- Identificar y comparar opciones para desarrollo del software de dispositivos embedded que permitan reducir plazos de diseño.
- Implementar conectividad con nodo central a través de módulo de conexión móvil GSM.
- Implementar módulo GPS para geoposicionamiento en caso de emergencia o necesidad.
- Desarrollar lado de aplicación con limitaciones, implementando intercambio de mensajes y comandos entre NCG y DMPs y viceversa. (No se implementa analítica de datos).
- Realizar análisis financiero de costes de prototipado.

1.5 Enfoque y método seguido

La realización de este proyecto conlleva la utilización de tecnologías, tanto de tipo hardware como software, no empleadas hasta ahora por el autor lo que exige una consideración especial, tanto en la planificación de riesgos a lo largo de las diferentes etapas como en la metodología empleada.

Desde la perspectiva de hardware, se emplearán interconexión de diferentes componentes: dispositivo base, módulo GPS, módulo GSM, estación de trabajo.

Para facilitar la incorporación al proyecto final se procederá mediante un esquema de pruebas iniciales de programación para cada componente a fin de validar su funcionamiento y obtener experiencia sobre su programación, a fin de integrarlo en el proyecto final de una forma más segura, reduciendo de este modo riesgos ante eventuales problemas en el código del proyecto final.

De este modo, en caso de encontrar dificultades en la prueba de algún componente resultará más sencillo depurar y resolver las incidencias en código fuente más reducido, en lugar de realizarlo en el proyecto final.

Por tanto, una vez seleccionadas las tecnologías hardware y software a emplear, para cada una de ellas se seguirá una metodología en esquema *Formación-Prueba inicial-Desarrollo.*

Ilustración 3:Metodología

1.6 Planificación del Trabajo

La planificación del proyecto sigue una estructura dividida siete ramas. Se orientan las cuatro primeras principales hacia la preparación previa al desarrollo del proyecto final, dos posteriores al desarrollo propio del proyecto y pruebas y una final orientada al cierre del proyecto.

A continuación se describen brevemente características esenciales de cada una de ellas:

1. Conceptualización: En esta fase se realiza una revisión del proyecto anterior que este pretende completar. Tras esto, se realiza un análisis de las mejoras aplicables al trabajo ya hecho mediante modificación, y aquellas aplicables mediante adición de nuevos componentes(GSM, GPS, IoT Cloud, App lado servidor)

2. Estado del arte: En esta fase se realiza un listado de productos alternativos existentes al presente proyecto. Para cada uno de ellos se realizará un análisis desde la perspectiva de competitividad y en caso que proceda de integración.

3. Selección de tecnologías: En esta fase se realizará un listado y análisis de tecnologías disponibles tanto de tipo hardware como software para el desarrollo del proyecto, analizando tanto aquellas que sirvan como herramientas de desarrollo como aquellas que sirvan a la puesta en producción del servicio.

4. Preparación pre-desarrollo: En esta fase se realiza la compra de materiales necesarios, registro en plataformas online para desarrollo. Tras la recepción de materiales, se procederá a la realización de primeras pruebas de programación cada componente con pequeños

códigos(ejemplos, GitHub, propios) para finalmente interconectar todos los dispositivos y realizar una prueba integral con código de test propio.

5. Desarrollo: Consiste en la fase de desarrollo propiamente dicha. La metodología seguida consistirá en el desarrollo modular del código tanto de dispositivo como del lado de aplicación. Finalmente se procede a la integración del código final.

6. Test: Se realizarán pruebas en mesa de trabajo. Finalmente se comprobará el carácter wereable del dispositivo realizando pruebas en exteriores durante un tiempo de test continuo.

7. Cierre: Se procederá al análisis de posibles trabajos futuros continuando la línea de desarrollo del proyecto. El trabajo realizado en esta fase no consiste en un análisis de posibles mejoras a aplicar sobre el dispositivo DMP, sino que observará el sistema de forma global considerando la totalidad de la arquitectura (DMP, Cloud IoT, NCG). En este sentido, se comentarán, entre otras, las posibilidades de implementar un agente conversacional artificial, que pueda cerrar el lazo de comunicaciones entre el DMP y el NCG mediante conversaciones telefónicas con el usuario a través del módulo GSM.

   A continuación se muestran las diferentes fases enunciadas en la lista anterior:

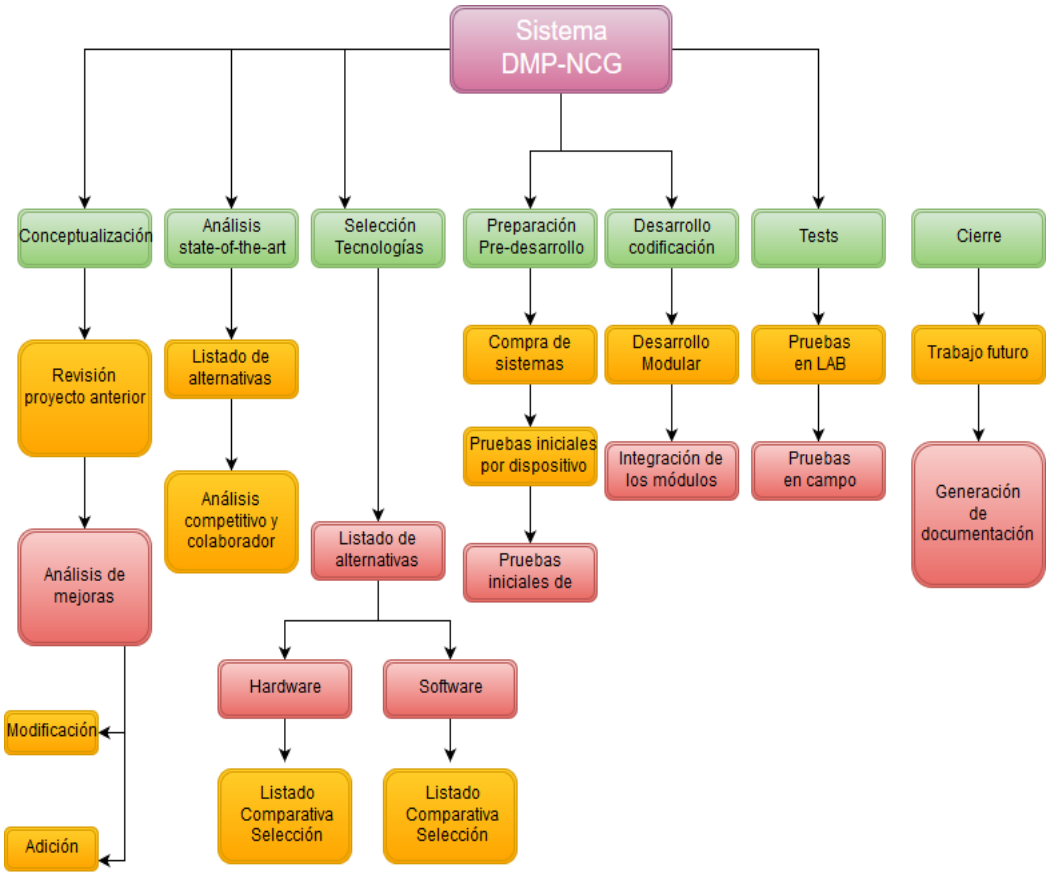


Ilustración 4:Jerarquía de proyecto

Para la realización de las diferentes fases se ha desarrollado una planificación temporal adecuada, asumiendo la existencia de riesgos potenciales debidos fundamentalmente a posibles fallos en recepción de materiales, averías potenciales de los mismos o la propia inexperiencia en las tecnologías a emplear.

En este sentido, se ha optado por plazos amplios que permitan flexibilidad en fechas deadline, de modo que el curso del proyecto se pueda reajustar, adaptándose a complicaciones que puedan surgir, incrementando de este modo las probabilidades de supervivencia y éxito.

Desde una perspectiva meramente temporal, se ha elaborado un plan cuyo diagrama se observa a continuación.

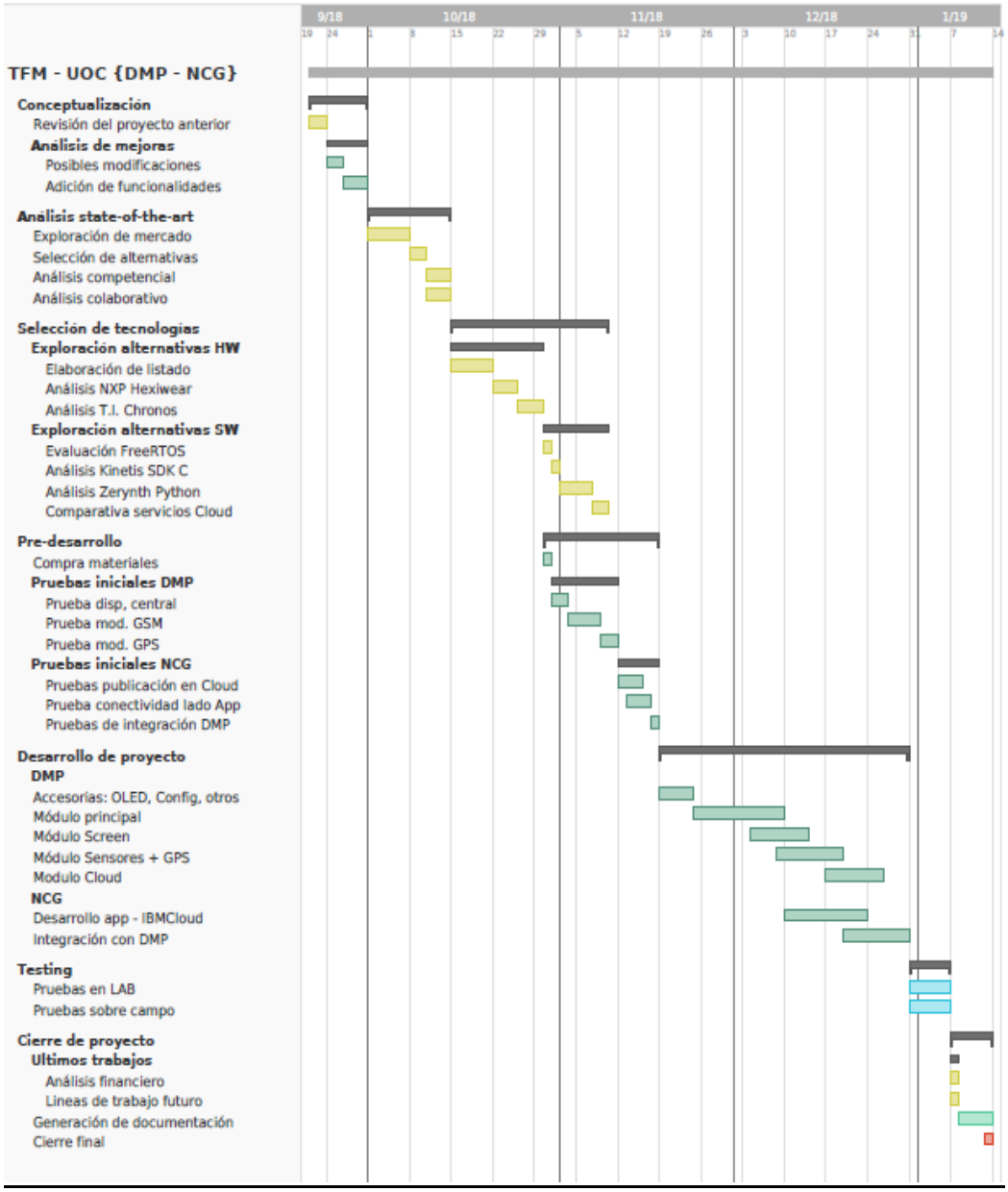


Ilustración 5:Planificación temporal

1.7 Breve sumario de productos obtenidos

Conseguidos los objetivos principales señalados en secciones anteriores, se obtienen dos productos principales:

- Implementación del DMP: empleando las tecnologías de desarrollo de dispositivos embeded indicadas en capítulos posteriores, de desarrolla y obtiene como producto un dispositivo DMPs con las funcionalidades pedidas. La implementación de dicho dispositivo se realiza sobre un Workstation que facilita la conexión de distintos módulos que permiten la implementación del conjunto de funcionalidades, tanto sensoras, como de comunicaciones y posicionamiento.

Ilustración 6: Implementación de DMP

- Implementación del NCG: El nodo central general se implementa sobre un servicio cloud que se encarga del almacenamiento de los datos publicados por los DMPs, además de la implementación de protocolos (MQTT) para la gestión y transmisión de mensajes de comando y publicaciones entre NCG y DMPs. En este sentido, se obtiene como producto entregable una pequeña aplicación escrita en lenguaje Python que implementa el lado servidor. La aplicación accede al servicio cloud mediante API key e intercambia mensajes con los DMP. Tras la conexión de un DMP, la aplicación le envía información de configuración y

posteriormente permite realizar el envío de mensajes simulando el frontend indicado en secciones anteriores.

Además se obtiene un código en Python que permite la simulación de DMPs para facilitar el desarrollo y permitir la comprobación del funcionamiento del NCG sin requerir ningún DMP físico conectado.

Ilustración 7:Notebook Jupiter para NCG

1.8 Breve descripción de los otros capítulos de la memoria

A continuación, se muestra, a lo largo de los próximos capítulos el camino recorrido durante la realización del proyecto.

En el capítulo siguiente, se realiza una breve exposición del estado del arte en el ámbito objetivo del proyecto, alternativas existentes y se profundizará en un proyecto concreto, Activage, debido a su gran alcance, inversión, y volumen de agentes implicados tanto públicos como privados.

En el capítulo 3, se realiza una exposición de tecnologías empleables para el desarrollo del sistema desde una perspectiva de hardware-software.

Finalmente en el capítulo 4 se expondrá la solución propuesta y realizada, tanto desde perspectiva de hardware como software, funcional como estructural.

En los últimos capítulos se expondrán líneas de trabajo futuro, no solo referidas al propio DMP, sino a toda la arquitectura, introduciendo otras tecnologías como machine-learning e IA.

2 Análisis previo

En el capítulo anterior se ha definido el problema existente y se han definido una metodología y planificación para alcanzar una solución que constituirá el producto de este trabajo.

Cabe por tanto realizar a continuación un estudio previo de las tecnologías existentes que por sus características y propósito se sitúen dentro del ámbito de actuación de este proyecto.

2.1 Estado actual de la tecnología IoT wereables

A lo largo de los últimos años, el sector de dispositivos conocidos como tecnologías de Internet de las cosas o IoT, ha venido experimentando un gran desarrollo tanto en investigación como en el desarrollo y aparición de dispositivos comerciales.

Tras la aparición de los teléfonos inteligentes en 2007, surgen los dispositivos conocidos comunmente como vestibles o wereables. Estos dispositivos particulares permiten la definición de las llamadas redes de área personal o Body Area Network(BAN) [BAN] que aplicadas a la monitorización del comportamiento humano permiten la obtención de datos útiles para tareas, como el ya tan extendido seguimiento de la actividad física en el desempeño de actividades deportivas.

La progresiva reducción de costes y la constante innovación en este sector permite la existencia de productos asequibles al usuario medio y con grandes capacidades y funcionalidades.

Desde un punto de vista técnico, referido al tipo de producto que se plantea, es decir, pulsera telemática, se observa la existencia de gran cantidad de productos con capacidad sensora y conectividad con teléfonos inteligentes que actúan como nodo de comunicación, permitiendo el almacenamiento de los datos adquiridos en servicios cloud.

Debido a la disrupción provocada recientemente por la incorporación de tecnologías de inteligencia artificial, se abren nuevas posibilidades al

aplicar estas tecnologías sobre los perfiles y datos almacenados. Si bien diferentes iniciativas ejemplifican esto, cabe destacar el proyecto Activage, que por su magnitud, fuerzas implicadas y ámbito europeo constituye un caso a analizar en mayor detalle, tal y como se realizará en la sección 2.3

Ilustración 8:Logo proyecto Activage

Por otro lado, la reducción de costes, y la democratización de tecnologías IoT, amplían el abanico de posibilidades y aunque no se propone en este proyecto, la redundancia de sensores no es prohibitiva en términos presupuestarios.

En esta linea, existen en el ámbito investigador diversas iniciativas que analizan las posibilidades de implementaciones de BAN constituidas por múltiples sensores vestibles, distribuidos en diferentes partes del cuerpo del usuario.

De este modo, la distribución de diversos sensores en unidades MIMU constituidas por acelerómetro, magnetómetro y giroscopio en las cuatro extremidades del usuario permite una monitorización exhaustiva de su comportamiento y actividad.

Este ejemplo descrito, arroja nuevas posibilidades no solo en lo que a la cantidad y calidad de datos adquiridos, sino también en lo que a la continuidad de servicio del sistema se refiere. La redundancia de sensores permite la generación de ontologías que facilitan la sustitución en servicio de sensores averiados por otros que generen información sustitutiva de calidad tal y como se analiza en [CLA].

2.2 Barreras de entrada: alternativas o productos similares

En el capítulo un se plantea como posible solución al problema un dispositivo vestible a modo de pulsera capaz de monitorizar la actividad

del usuario, generar alarmas en ciertas situaciones de peligro o necesidad y con conectividad móvil para la transmisión de éstas y de los datos adquiridos.

Cabe recordar, tal y como se especifica en el capítulo anterior, que el escenario del problema es muy concreto, ya que se sitúa en el rural gallego, y esto sugiere la necesidad de análisis de las condiciones específicas del problema.

Cuestiones la orografía, la climatología, son aspectos a análizar para este caso particular y pueden indicar la necesidad de consideraciones especiales que sugieran un dispositivo y sistema diseñado a medida del problema particular.

Sin duda otra cuestión a tener en cuenta es la cobertura móvil disponible el escenario rural que se pretende resolver no dispone de cobertura plena. A continuación se muestran diversos pueblos objetivo del proyecto sin cobertura básica 2G para el operador mayoritario Movistar.

Ilustración 9:Ejemplo Cobertura en zona rural Galicia

En el mercado existen opciones generalistas y otras más particulares, con diseños, funciones y precios orientados a un cliente unipersonal con requerimientos generales. Otras soluciones se orientan hacia objetivos sanitarios y de prevención de enfermedades.

Sin embargo la existencia de proyectos públicos de gran presupuesto, orientados al sector sanitario parece señalar faltas en la atención de empresas privadas a este ámbito.

Si bien, el alcance efectivo del proyecto es mayor abarcando otros objetivos tanto principales como secundarios, el principal entre todos ellos consiste en resolver el problema de personas errantes en el rural gallego, que tal y como ya se ha explicado, concluye en elevado numero de ocasiones, con desenlace fatal para la persona.

En cumplimiento de este objetivo, el primer producto rival o alternativa a la solución propuesta es el desarrollo de una aplicación para smartphone capaz de acceder a sus módulos de conectividad y geoposicionamiento y cuyo enfoque, sea principalmente el público objetivo y sus circunstancias especiales: gente de avanzada edad en entorno rural.

No obstante esta solución no es adecuada. La primera consideración a tener en cuenta la constituyen las características del público objetivo. Se trata de personas de edad avanzada, con niveles de cultura tecnológica bajos que además pueden presentar menoscabo en funciones motoras, o sensoras como por ejemplo una disminución en su capacidad sensora visual.

Esto puede concluir en una inoperancia del sistema, sirva como ejemplo el resurgimiento de móviles sin pantalla táctil y con teclado convencional dirigidos precisamente a este tipo de público objetivo.

Otro aspecto a tener en cuenta lo constituye la eficiencia del sistema entendida desde la perspectiva de su adecuación al problema propuesto.

En este sentido, el uso de la batería del dispositivo resultará totalmente ineficiente, ya que se consumirá para otras utilidades propias del dispositivo en lugar de priorizarse en diseño y políticas de rendimiento hacia el objetivo principal.

Finalmente, en lo relativo a este tipo de solución, cabe citar lo reseñado en el trabajo anterior [CES] al que este proyecto pretende dar continuación, en el que ya se mencionaba la existencia de un servicio de pago basado en teléfonos inteligentes provisto por Cruz Roja.

Actualmente Cruz Roja en Galicia oferta una modalidad básica de teleasistencia para personas de edad avanzada que viven solas y que consiste en un único pulsador llevado al cuello a modo de colgante conectado de forma inalámbrica con un receptor enlazado a la linea telefónica.

Ilustración 10:Servicio teleasistencia Cruz Roja

Adicionalmente Cruz Roja proporciona un servicio de teleasistencia más completo basado en teléfonos inteligentes que explota las funcionalidades extra que estos dispositivos proporcionan.

Sin embargo, Cruz Roja se ha incorporado al Proyecto Activage que se describirá en siguientes secciones buscando soluciones a medida y por tanto más eficientes.

Examinando otros productos orientados a cliente final, existen pulseras comerciales tales como FitBit o la banda de Xiaomi:

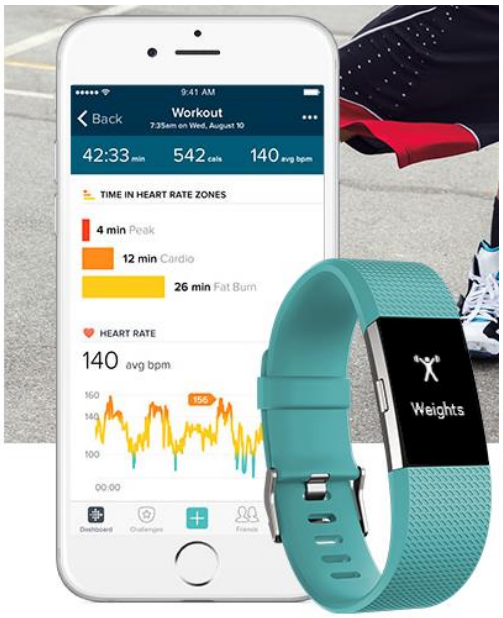


Ilustración 11:Pulsera FitBit Charge 2

Modelos como el FitBit Charge 2 presentan un gran número de funcionalidades:

- Monitorización de la actividad mediante multisensores.
- Conectividad GPS continua.
- SmartTrack: Memorización de actividades de desplazamiento como caminatas, carrera continua o bicicleta.

Sin embargo, tal y como se indicó en secciones anteriores, servicios como la conectividad y el geoposicionamiento exigen su sincronización con un teléfono inteligente con las desventajas correspondientes ya señaladas.

Otra alternativa la componen dispositivos más complejos como los relojes inteligentes o smartwatch como el desarrollado por Apple:

Ilustración 12:Apple Smartwatch series 4

El Apple Smartwatch series 4 incorpora un extraordinario conjunto de funcionalidades que incluye todas las referidas a monitoreo de la actividad, deporte, reproducción de música, pantalla táctil, etc.

Además este modelo incorpora un dispositivo GPS integrado que proporciona geoposicionamiento. También incluye conectividad móvil y es capaz de realizar y recibir llamadas telefónicas con lo que consiste prácticamente en un teléfono inteligente miniaturizado en una pulsera.

Además es posible desarrollar aplicaciones a medida para el smartwatch con lo que se podrían implementar las funcionalidades exigidas para realizar un producto casi a medida.

Sin embargo también se sitúa en clara desventaja frente a un producto realizado a medida. Además de encontrar las mismas barreras de uso por parte del público objetivo que el caso del teléfono inteligente, su corta duración de batería y principalmente su elevado coste, a partir de 429 euros, lo sitúa fuera de las soluciones posibles.

Una vez considerados los productos destinados a mercado generalista, debe analizarse la existencia de productos destinados a funciones directamente relacionadas con este proyecto. Entre ellos, existen dispositivos alternativas como el producto Nock Senior, que consiste en un reloj de pulsera con capacidades con capacidad GPS y conectividad móvil.

Ilustración 13:: Dispositivo Nock

Este dispositivo sí que se centra el en publico objetivo de este proyecto, gente de edad, proporciona capacidad de recibir llamadas y un botón de pánico. La comercialización del sistema funciona por suscripción mensual a 12 euros/mes, trimestral por 30 euros/mes, o anual por 99 euros al año.

Sin embargo la solución propuesta por este proyecto pretende una gestión del geoposicionamiento y uso de comunicaciones móviles a

medida para el escenario rural a resolver, además de proporcionar sensorización extra que permita la generación de alarmas.

Otra alternativa existente en el mercado es la proporcionada por Vodafone denominada V-SOS Band.

Ilustración 14:: V-SOS Band

Este dispositivo sí que proporciona además del geoposicionamiento, conectividad móvil y sensor de caidas. El servicio supone un desembolso de 79 euros más 5 euros en concepto de suscripción mensual.

En contraposición, el sistema propuesto incluye mayor sensorización, recopilación de información útil, envío preventivo de posición ante amenaza de pérdida de cobertura móvil. Por otro lado el servicio de V-SOS Band funciona únicamente bajo cobertura Vodafone mientras que el propuesto admite tarjeta SIM de cualquier operador, pudiendo elegir el más adecuado en función de la cobertura disponible en la residencia habitual del usuario.

2.3 Proyecto Europeo ACTIVAGE

En la sección anterior se ha realizado una recopilación de posibles soluciones existentes en el mercado comenzando por sistemas y dispositivos comerciales dirigidos a un público general, continuando con productos dirigidos al ámbito que pretende cubrir este proyecto.

Sin embargo, no parecen existir productos de origen privado dirigidos al sector que verdaderamente resuelvan el problema, al menos a juicio del ente público.

Constituye evidencia de ello la existencia de un proyecto como es Activage, http://www.activageproject.eu/, tanto por su ambición de producto, por la implicación tan numerosa de agentes diversos públicos y privados, por su magnitud presupuestaria y por su amplio objetivo geográfico a nivel europeo.

El proyecto Activage se activa en 2017 con un plazo de desarrollo de 42 meses encontrándose en el bienio 2018-2020 en fase de pruebas piloto. El proyecto ofrecerá como productos dispositivos y sistemas además de herramientas y metodologías, garantizando la interoperabilidad de sistemas con el objetivo final de favorecer el envejecimiento saludable de la población incrementando el grado de independencia de los ancianos.

El proyeto aglutina agentes de la industria, centros de investigación, y entes públicos de carácter internacional.

Ilustración 15:Entidades Partners de Activage

El proyecto dispone de 20 millones de euros de financiación, además dispone de un concurso Open Call a través del cual seleccionará y financiará con 600.000 euros a diez empresas cuyos esfuerzos estén orientados a este ámbito.

El proyecto se encuentra en pruebas piloto en 7 países, a través de 9 despliegues o Deploy Sites.

En España se encuentran tres de estos DS siendo Galicia uno de ellos.

2.3.1 Galicia como Deploy Site

En Galicia forman como asociados en el proyecto las siguientes entidades:

- Televés S.A.
- Cruz Roja Española
- Servicio Galego de Saude
- Universitat Politécnica de Valencia
- Fundación Vodafone

Además se emplean tecnologías como Sofia 2 de Indra o OpenIoT.

La prueba piloto cuenta con aproximadamente 700 usuarios mayores de 60 años, con alguna característica que pueda limitar su independencia: enfermedad crónica, baja actividad social, diabetes, etc.

El piloto consiste en la ubicación en el domicilio del usuario de un nodo de comunicaciones desarrollado por Televés S.A. que se encarga de recopilar datos de los diferentes sensores distribuidos por la vivienda , en puertas, electrodomésticos, cama, etc. que permiten monitorizar la situación y actividad del usuario para remitirlos al servicio en la nube y a la familia del usuario. Por su parte Cruz Roja proporciona asistencia a domicilio presencial y también a través de enlace telefónico con los usuarios.

3 Plataformas y tecnologías para prototipado rápido embedded

Tal y como se ha mencionado en apartados del capítulo inicial, este proyecto deriva de otro realizado por el autor anteriormente. En dicho proyecto se realizó en su momento un breve estudio de alternativas hardware y software para el desarrollo de la prueba de concepto propuesta. Este análisis concluyó en la utilización de la placa de evaluación LPCxpresso basada en el micro LPC1679 del fabricante NXP, además de diversos dispositivos externos conectados a la misma mediante una placa de prototipado convencional.

Sin embargo, LPCxpresso es una placa de propósito generalista de gran tamaño y aunque válida para tal prueba de concepto, no resulta apropiada dados los objetivos de este proyecto expresados en el capítulo 1.

Para la realización de este proyecto, tal y como se ha comentado anteriormente, se pretende entre otros objetivos, lograr un prototipo que permita implementar las funcionalidades del dispositivo como *wereable* de IoT.

Para ello se ha realizado una búsqueda de diferentes opciones existentes en mercado, en materia de placas o dispositivos de prototipado rápido orientado a dispositivos vestibles o *wereables*.

Producto de dicha búsqueda se presentan dos alternativas con mayor peso. Por un lado el producto Hexiwear desarrollado por la empresa Mikroelectronika en partnership con el fabricante de dispositivos NXP. Por otro lado se cuenta con el dispositivo de pulsera Chronos del fabricante Texas Instruments.

De ambos se muestra una descripción desde perspectiva hardware y software en las dos siguientes secciones.

3.1 Plataformas Hardware

3.1.1 Hexiwear

Desde una perspectiva puramente de hardware, el dispositivo Hexiwear constituye un punto de partida excepcional para prototipado rápido de dispositivos wereables en general y de pulsera en particular.

En el capítulo uno se había presentado el proyecto anterior que da origen a este, y se observó la dificultad de desarrollar un prototipo adecuado para testing debido al tamaño excesivo del mismo, al tener que emplear una placa de prototipado para implementar sensores y dispositivos de forma externa.

Este problema viene resuelto de antemano por Hexiwear, que incorpora en una misma placa de reducido tamaño gran parte de los sensores y dispositivos periféricos deseables en aplicaciones del sector IoT personales.

El dispositivo se presenta en un tamaño adecuado para aplicaciones de tipo pulsera aunque es también indicado para otras localizaciones al poder ser accesoria y removible.

Ilustración 16:Hexiwear y accesorios

En una vista exterior, el dispositivo se muestra del tamaño aproximado de un reloj convencional, con una pantalla OLED de 96x96 pixels, un puerto USB y seis pulsadores por contacto de tipo capacitivo.

Además se incorpora en el paquete de compra dos sujeciones diferentes e intercambiables, una a modo de pulsera y otra a modo de colgante y embellecedores de diferentes colores también intercambiables.

Es destacable que los diseños en 3d en formato STL de las partes rígidas mencionadas hasta ahora como las sujeciones, pulsera y colgante, como carcasa del dispositivo son accesibles con licencia gratuita y modificables mediante programas especializados en diseño 3d como el gratuito FreeCad.

Profundizando en el interior del dispositivo se comprueba gran grado de personalización con vistas a prototipado. No se trata de un dispositivo sellado, sino que puede desmontarse separando partes de carcasa de la placa electrónica interior, batería y pantalla.

Ilustración 17:Desmontaje del dispositivo

Descendiendo en el diseño y aumentando en el nivel de detalle se alcanza la placa principal del dispositivo portante de gran número de sensores onboard.

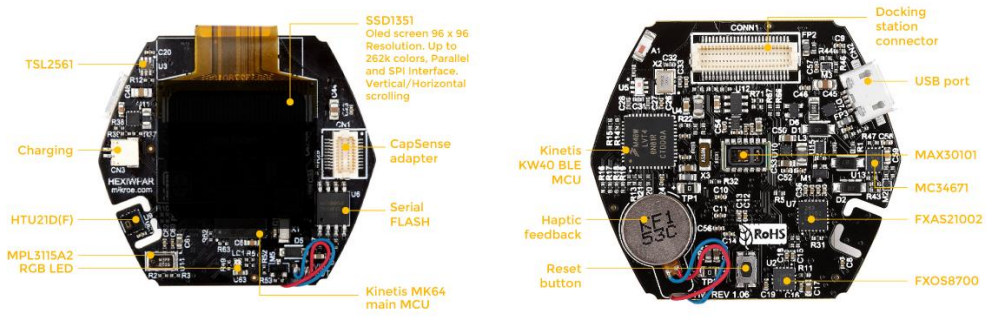


Ilustración 18:Placa principal de Hexiwear

Analizando más en detalle la composición de la placa, Hexiwear monta dos microcontroladores de la familia Kinetis desarrollados por el fabricante NXP.

El microcontrolador Kinetis KW40Z es empleado para gestionar la comunicación vía Bluetooth de baja consumo o BLE del dispositivo Hexiwear con otros dispositivos como smartphones.

Se trata de un microcontrolador de la famila ARM Cortex M0 con las siguientes características:

- Memoria: 160 KB de Flash y 20 KB de RAM
- Radio BLE v1.4
- Comunicaciones: Vía UART y SPI.

En segundo lugar, el dispositivo Hexiwear monta un microcontrolador Kinetis MK64 como microcontrolador principal para el gobierno del dispositivo. Se trata de un microcontrolador de la familia ARM Cortex M4 con las siguientes características:

- Microcontrolador de bajo consumo(Ultra-Low-Power)
- ARM Cortex M4 con DSP
- DMA de 16 canales
- Timer para gestión de interrupciones y conversiones ADC de 32 bits
- 1 MB de Flash
- 256 KB de RAM
- 8 Pins Analógicos
- Reloj 120 MHz
- 76 Pins I/O
- FlexMemory 4KB

- 2 ADCs de 16 bits de alta velocidad con tiempos de conversión de 500 ns
- 2 DACs de 12 bits
- 6 UARTs
- 3 I2C
- 3 SPI

Cuyo diagrama de bloques se muestra en la figura siguiente:

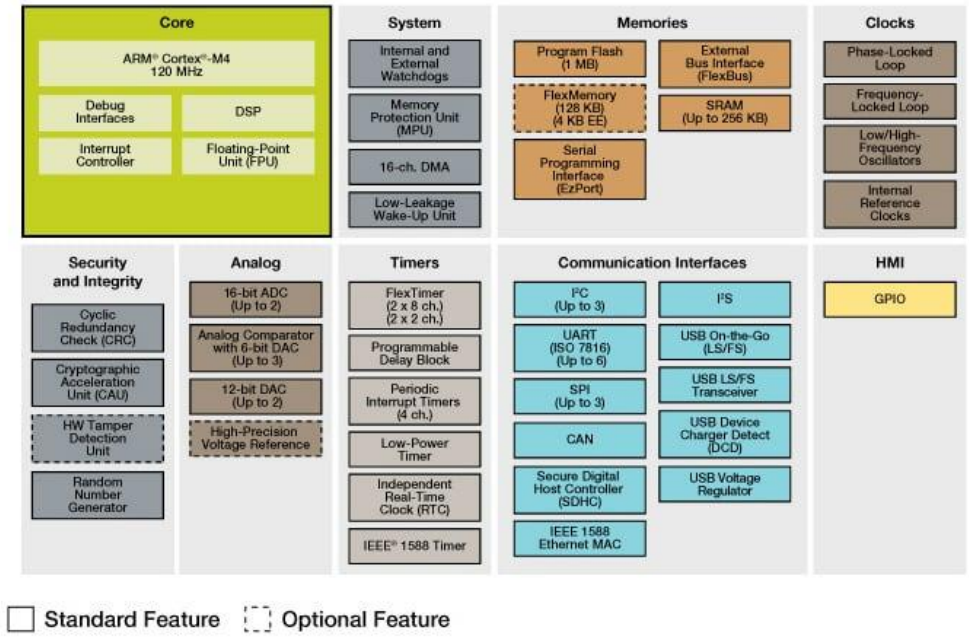


Ilustración 19:Diagrama de bloques Kinetis MK64

Por lo que respecta al resto de componentes de la placa se observan los siguientes dispositivos:

- MIMU 3D (acelerómetro, magnetómetro) de 6 ejes: NXP FXOS8700CQ
- Giroscopio del fabricante NXP FXAS21002CQ
- Sensor de presión de NXP MPL3115A2R1
- Cargador de batería 600mA de tipo Li-Ion NXP MC34671
- Batería de tipo LiPo de 190 mAh

- Sensor de intensidad lumínica
- Sensor de temperatura y humedad HTU21D
- Sensor de pulso cardíaco MAX30101
- 1 Led tricolor RGB
- Puerto microUSB
- Pequeño motor de vibración

El conjunto de componentes y dispositivos descritos se articulan para conformar Hexiwear según el siguiente diagrama de bloques del fabricante NXP.

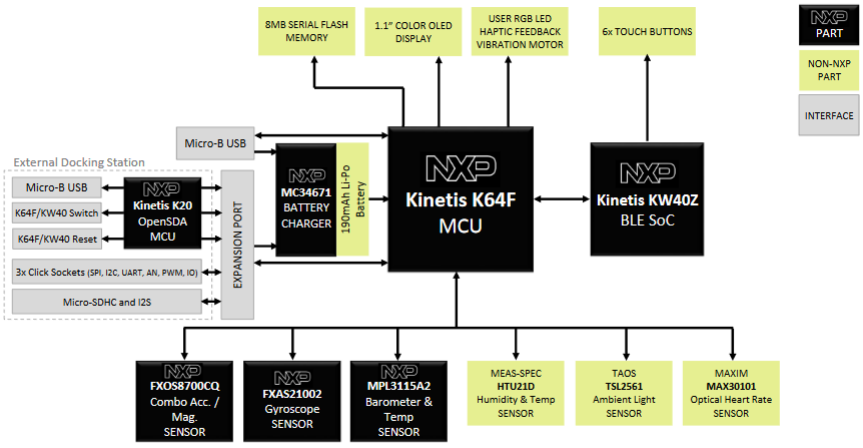


Ilustración 20:Diagrama bloques Hexiwear

A la vista de la gran potencia del dispositivo Hexiwear y de su versatilidad permitida por el gran número de sensores montados onboard, así como las capacidades en comunicaciones el dispositivo se postula como un gran candidato para el desarrollo del proyecto.

Sin embargo existen ciertas limitaciones que serán comentadas en capítulos posteriores.

Un caso claro de ellas se ve representado en el sensor de pulso cardíaco que tal y como se comentará en capítulos posteriores presenta una operatividad muy baja debido a su falta de precisión.

Otro ejemplo se refiere principalmente a la corta duración de la batería montada por el dispositivo, más aún teniendo en cuenta que se pretende incorporar comunicaciones vía GSM/GPRS lo que incrementará notablemente los costes energéticos del funcionamiento.

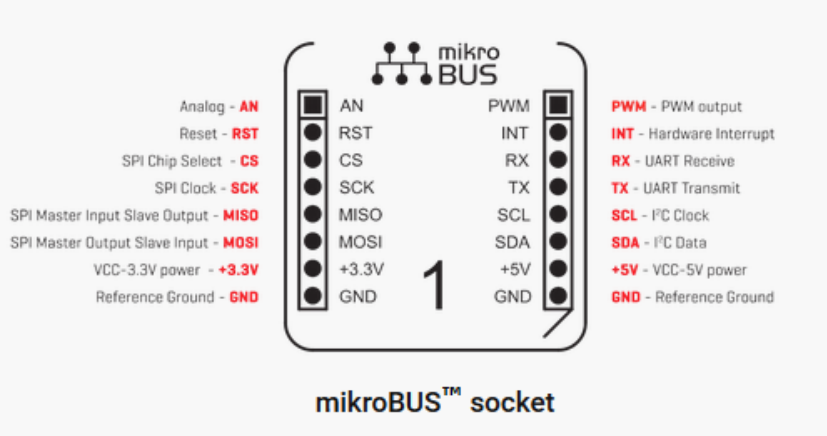
No obstante, también en capítulos posteriores se planteará solución a dichas limitaciones como parte del proyecto.

Tal y como se ha comentado en esta sección, el dispositivo Hexiwear incorpora comunicaciones Bluetooth de baja energía o BLE. Esto permite enlazar el dispositivo con un smartphone y a través de una aplicación desarrollada por Mikroelectronika para Hexiwear, monitorizar sensores e incluso enviar datos a un servicio cloud.

Sin embargo, este proyecto, tal y como se requiere según las especificaciones expresadas en el capítulo 1, requiere independencia de otros dispositivos, por tanto se pretende desarrollar un prototipo autónomo capaz de realizar el envío de datos al nodo central o NCG sin requerir dispositivos auxiliares. Además se exige geoposicionamiento GPS, cosa que Hexiwear no es capaz de proveer.

Por tanto cobra vital importancia la capacidad de expansión del dispositivo y esta se realiza a través del denominao MikroBus.

MikroBus es una tecnología desarrollada por la empresa diseñadora de Hexiwear que permite la expansión de placas de evaluación mediante la adición de los denominados clicks de expansión.



mikroBUS™ socket

Ilustración 21:Esquema de MikroBus

La empresa Mikroelectrónica a través de su página web proporciona los diseños esquemáticos y el diseño PCB de MikroBus. Si bien se trata de una marca registrada, la licencia permite a cualquier diseñador la utilización de este sistema en proyectos propios bajo premisa de cumplir los requerimientos del estándar de diseño definido por la empresa.

El conector MikroBus representado en la figura anterior muestra una estructura de dos líneas de 8 pines hembra distribuidas en diversos grupos según su función. Se observan tres grupos de comunicaciones que integran UART, I2C y SPI, además de pines para gestión de interrupciones, PWM, reset, chipset y entrada analógica.

Además se ofrecen dos posibilidades de alimentación del click de expansión, bien 3,3 V o 5 V para lo que se habilitan cuatro pines incluyendo dos para GND.

La versatilidad que proporciona las posibilidades de expansión mediante MikroBus se ve reflejada en su adopción por parte de terceros fabricantes como Microchip, ATmega, Infineon o incluso NXP en su placa i.Mx7 Dual SABRE board.

La empresa Mikroelectronica dispone de un amplio catálogo de clicks de ampliación de diseño propio, que alcanza a fecha de 2018 la cifra de 562 y que constituyen placas de evaluación de diversos dispositivos existentes en el mercado.

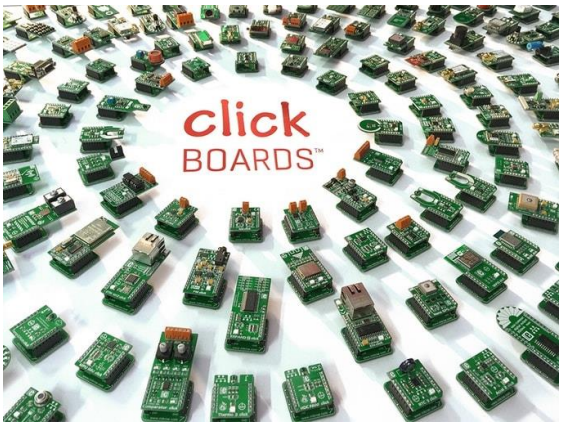


Ilustración 22:Clicks de expansión de Hexiwear

Por tanto, si se desea incorporar una funcionalidad nueva al dispositivo hexiwear basta con seleccionar un click del catálogo que la incorpore y conectarlo al dispositivo Hexiwear vía MikroBus.

La conexión de clicks con el dispositivo Hexiwear se realiza mediante una estación de trabajo o *dock-station*:

Ilustración 23: dock-station de Hexiwear

Esta estación de trabajo incorpora tres conectores MikroBus que permiten la utilización de otros tantos clicks de expansión. Además de otras funcionalidades que se citan a continuación:

- Slot para tarjeta MicroSD.
- 3 LEDs informativos, uno por cada conector MikroBus
- Interruptor de arrancado
- Puerto MicroUSB para alimentación y programación.
- Interfaz vía OpenSDA con 8 conmutadores binarios. Su configuración permite programación vía micro MicroUSB o mediante programador externo. También permiten seleccionar el

micro objetivo a programar entre los dos Kinetis disponibles, si bien el MK64 o el de comunicación BLE MKW40.

- Dos botones pulsadores de reinicio. Uno para cada microcontrolador Kinetis que incorpora Hexiwear.
- Conectores JTAG. Permiten programación de Hexiwear mediante un programador externo.
- Interfaz I2S

En caso de requerir mayores funcionalidades por parte de la estación de trabajo, otra opción consiste en la placa de trabajo Hexiwear WorkStation.

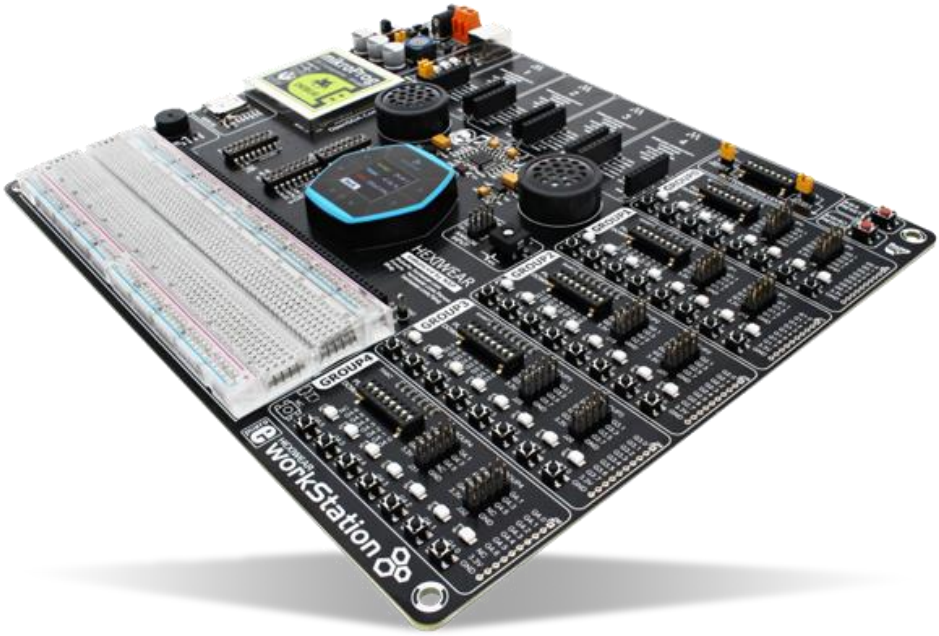


Ilustración 24:WorkStation de Hexiwear

3.2 Tecnologías de software

3.2.1 Zerynth y Python para diseño embebido

Tras explorar las soluciones convencionales de herramientas de software para diseño de sistemas embedded basadas en lenguaje C, se analiza una alternativa relativamente joven aunque a su vez interesante.

Zerynth supone una solución integral como herramienta de diseño que basa su principal activo en la utilización de máquinas virtuales como capa de abstracción en su stack, de modo que al contrario que ocurre con otras soluciones específicas, como la descrita en la sección anterior, se implementa una cierta independencia entre el software a diseñar y el hardware sobre el que se pretende implementar.

Esto de por sí, implica la posibilidad de diseñar software para una aplicación determinada, pudiendo llegar a abstraerlo de futuros cambios en el hardware, por mejoras, ampliaciones u otros, incrementando la reusabilidad del código y sobre todo orientándolo en mayor medida hacia la aplicación y disminuyendo su dependencia de hardware.

Desde una perspectiva introductoria la solución Zerynth se basa en un IDE integral denominado Zerynth Studio de descarga gratuita desde la web de la compañía.

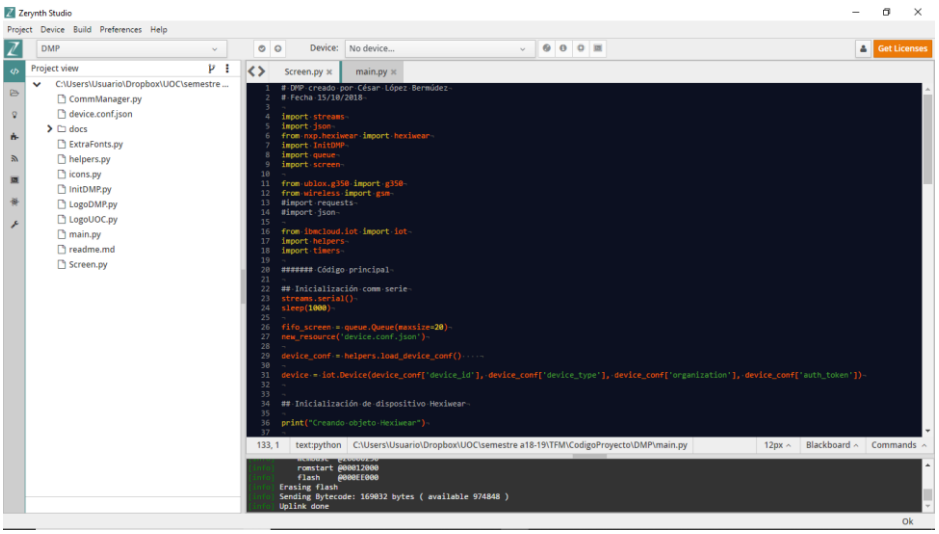


Ilustración 25:Vista del IDE Zerynth Studio

Zerynth Studio IDE se sitúa en la cima de la suit Zerynth Tool Chain, accesible también vía línea de comandos y permite el desarrollo de proyectos embedded, gestión y desarrollo de código, acceso a numerosos proyectos de ejemplo de uso de librerías y registro, virtualización y programación de hardware embedded.

Con la descarga del software se permiten un número limitado de licencias de virtualización, siendo requerida una licencia por cada dispositivo que se desee comercializar. Se reciben de inicio cinco licencias de tipo *estándar* y otras cinco de nivel *premium*. Todas incorporan el sistema operativo en tiempo real FreeRTOS aunque las licencias premium incorporan control y análisis de consumo energético, securización via hardware y actualización via FOTA.

Además Zerynth Studio dispone de integración con GIT, permitiendo almacenar los proyectos en GitHub.

El desarrollo de software se realiza en lenguaje Python para el que Zerynth proporciona tres grupos de librerías:

- Librería estándar: Conformada por los estándares de Python, librerías de cálculo matemático, gestión GPIO, periféricos DAC, ADC, SPI, I2C, etc…, algunos de los más relevantes para este proyecto se citan a continuación.

Referencia	Descripción
Streams	Control de transmisión serial
Threading	Gestión de ejecución multitarea(Tasks de FreeRTOS)
GSM	Gestión de conexión y comunicación sobre GSM
Queue	Gestión de colas de mensajes entre Threads en multitarea
JSON	Gestión de formato JSON
Timers	Gestión de temporizadores

Tabla 1:Librerías Standard de proyecto

- Librerías oficiales: Constituido por un gran conjunto de librerías que incluyen desde drivers y gestión de dispositivos de diversos fabricantes hasta gestión de protocolos como HTTPS, TLS o MQTT, e interacción con servicios en la nube como IBMCloud IoT, Google Cloud Platform, o AWS de Amazon, tecnologías BlockChain como Ethereum y hardware de diversos fabricantes. Algunas de ellas, de mayor relevancia para este proyecto se citan en la tabla siguiente:

Referencia	Descripción
NXP	Librerías para micros y dispositivos del fabricante incluyendo una integral para Hexiwear y su ecosistema
IbmCloud	Acceso, suscripción y publicación al servicio cloud de IBM.
Ublox	Driver y librería para el modem GSM/GPRS Sara G350
MQTT	Librería que implementa el protocolo MQTT para comunicación con servicios cloud.

Tabla 2:Librerías Oficiales de proyecto

- Librerías de Partners: Permiten interacción con servicios cloud como Wolkabout, LiveIntersect o Ubidots.
- Librerías desarrolladas por la comunidad: Desarrolladas por la comunidad de usuarios de Zerynth bajo supervisión y aprobación previa de la compañía. En la actualidad existen únicamente dos: Librería para control gestual basada en APDS-9960, y la librería para el SRF01(ultrasonidos)

Por tanto, Zerynth supone un ecosistema de desarrollo integral especialmente orientado a diseño y desarrollo en el sector IoT proporcionando cobertura sobre los grandes actores del sector:

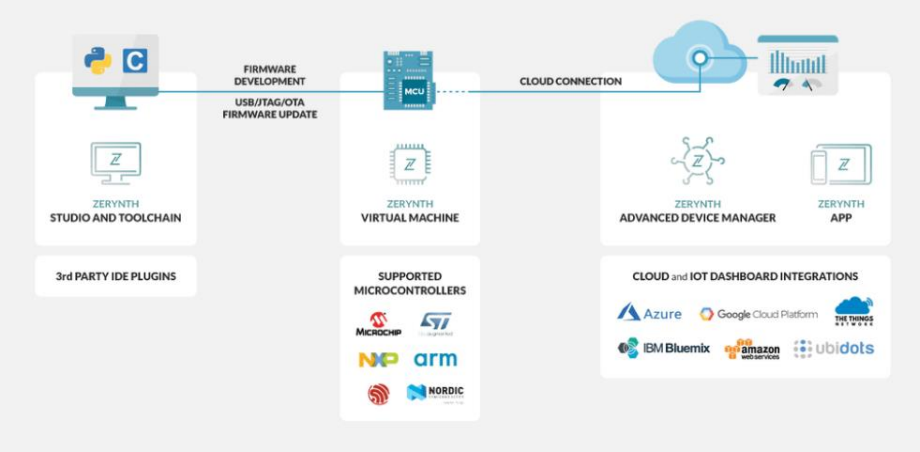


Ilustración 26: Ecosistema Zerynth

Aumentando el nivel de detalle del análisis se describe el *stack* de la herramienta articulado sobre el eje de la máquina virtual.

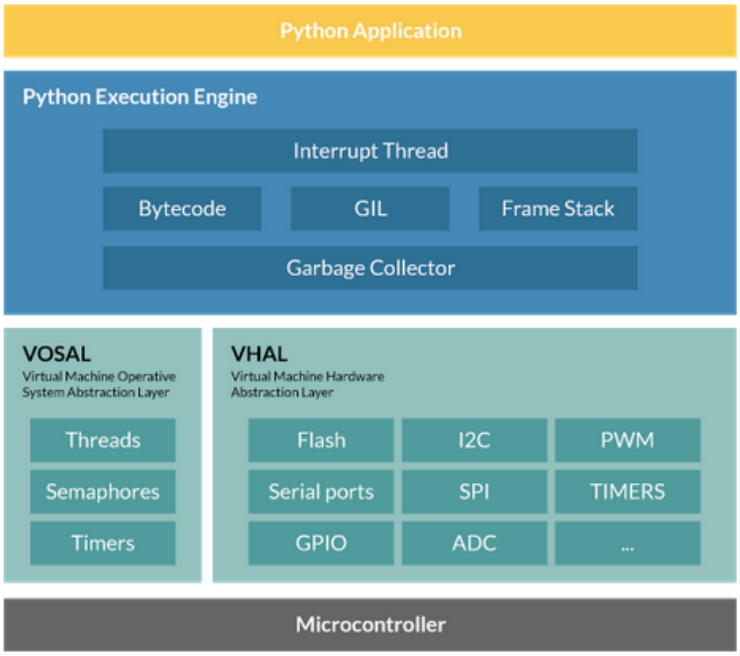


Ilustración 27:stack Zerynth

Tal y como se aprecia en la figura anterior, la aplicación desarrollada en Pythonc/C descansa sobre un stack en cuya base se encuentra el hardware elegido para el proyecto. Entre ambos se sitúan dos bloques que garantizan la independencia entre ambos extremos. Bajo la capa de aplicación se sitúa un motor de ejecución para Python cuyos componentes esenciales se describen a continuación. Bajo esta capa se sitúa una capa de abstracción compuesta por una máquina virtual compuesta por dos unidades, una encargada del aprovechamiento de las capacidades del sistema operativo elegido, capa VOSAL, y otra encargada del hardware, capa VHAL.

Según la propia documentación de Zerinth accesible desde su web o desde el IDE la máquina virtual ha sido creada partiendo de cero y como principales componentes se citan los siguientes:

- Bytecode interpreter: Se encarga de procesar los scripts de Python que componen el proyecto generando objetos bytecode compuestos de instrucciones denominadas *opcodes*.
- Global interpreter Lock o GIL: garantiza la atomicidad de los opcodes para proporcionar un entorno thread-safe en proyectos multitarea. De este modo se garantiza que ningún opcode de un determinado thread puede interrumpir a otro opcode de otro thread diferente, sino que la conmutación de ejecución se realizará cuando acabe el slot de tiempo asignado a cada thread o bien el propio thread finalice.
- Garbage collector: Representa una ayuda más de la herramienta al programador. Zerynth dispone de esta herramienta para remover automáticamente y de forma segura, aquellos objetos creados por el programador que ya han concluido su ciclo de vida, cosa que no ocurre con otras herramientas basadas en C de bajo nivel como las citadas en capítulos anteriores, es el propio programador quien se encarga de la gestión y liberación de memoria.

- Interrupt Thread: Se trata de un thread de ejecución al que se le otorga máxima prioridad a fin de que ejecute las funciones asignadas a cada interrupción cuando esta se active.
- Virtual Machine Operative System Abstraction Layer o VOSAL: Se trata de una capa de abstracción de sistema operativo, que puede ser ChibiOS o como se utilizará en este proyecto FreeRTOS. Esta capa incorpora funcionalidades para implementar las capacidades del sistema operativo multitask elegido, estas pueden ser accedidas directamente desde código de proyecto. Por tanto es la encargada proporcionar accesibilidad a semáforos, mutex, etc…
- Virtual Machine Hardware Abstraction Layer o VHAL: Esta capa de abstracción proporciona acceso y capacidad de control sobre el hardware disponible en el dispositivo a programar. VHAL define la gestión de puertos, SPI, I2D convertidores digitales y cualquier otro dispositivo de que pueda disponer el hardware.

Una ventaja más de la solución Zerynth es la posibilidad de programación híbrida en lo que a lenguaje se refiere. Si bien el lenguaje estándar de desarrollo es Python, Zerynth admite programación híbrida pudiendo combinar código en Python con código en C.

De este modo, es posible aprovechar las ventajas del uso de librerías y desarrollo rápido basado en Python, e implementar partes de código en C si fuese necesario a efectos de desarrollar un código óptimo y eficiente en caso de emplear hardware no contemplado en las librerías.

Quizá la mayor duda posible sea en cuanto a la posible eficiencia del código generado empleando Zerynth respecto a otros IDEs basados puramente en C como es el caso de Kinetis SDK.

Tal y como se ha especificado anteriormente, Zerynth no se ha desarrollado partiendo de versiones ligeras existentes de Python como microPython, sino que se ha desarrollado partiendo desde cero en busca de máxima eficiencia, logrando un footprint de máquina virtual bajo situado en un rango de 60-80 Kb y 5-6 Kb de RAM.

Sin embargo, sí que se aprecian ciertas pérdidas de eficiencia a la hora de ejecución de la capa de aplicación. Un análisis detallado de tiempos de ejecución y respuesta bajo FreeRTOS con aplicaciones desarrolladas en C puro confrontados con los tiempos de las mismas funcionalidades programadas mediante Python en Zerynth, arroja como resultado pérdidas de rendimiento en cuanto a cambios de contexto en ejecución y en la latencia de interrupciones.

Según se desprende del análisis realizado por Nisar Ahmad [PSA], se observa una latencia en interrupciones de 37,75 us para código Python desarrollado mediante Zerynth, mientras que para código C puro se reduce a tan solo 3.3 us. Por tanto la frecuencia máxima de activación de interrupciones no debe exceder los 26KHZ.

Por su parte, en el análisis de tiempos durante cambio de contexto realizado en el mismo trabajo [PSA], también arroja pérdidas de eficiencia, situándose el tiempo de cambio de contexto en 27.5 us para el caso de C frente a 160 us para el caso de Python en Zerynth.

A modo de conclusión, existe un compromiso claro entre eficiencia en tiempos de ejecución frente a las ventajas de portabilidad sobre hardware que ofrece Zerynth y disminución de los tiempos de desarrollo.

En este proyecto, habida cuenta de la capacidad suficiente del hardware elegido, se empleará esta herramienta para realizar el desarrollo, aprovechando las ventajas citadas a las que se une la utilización de un único lenguaje de programación para el desarrollo íntegro del proyecto, tanto a nivel de dispositivo DMP como en el lado de aplicación en servidor.

3.2.2 Plataformas Cloud para IoT

En la actualidad existen multitud de proveedores de servicio Cloud para IoT, no obstante en base a la decisión de optar por el ecosistema Zerynth como herramienta para el desarrollo del proyecto, han de

considerarse aquellos proveedores para los que Zerynth proporciona librería de conectividad.

Revisada la documentación de Zerynth, se observan diversas alternativas, dada la naturaleza académica del proyecto se considerarán especialmente los servicios de los planes gratuitos. Se procede a un pequeño análisis de tres de ellas:

- Wolkabout Cloud: En la primera prueba de programación del SARA G350 se ha realizado conexión con este servicio cloud.

  Se trata de un sistema open source de servicios cloud con un panel de control de fácil manejo para la visualización de datos publicados.

  Dispone de tres modalidades de pago:

  - Freemium: Supone coste cero. Tal y como se comentó anteriormente, este plan se eligió para una primera prueba de programación de G350 y MQTT. Sin embargo presenta una limitación que lo vuelve inoperativo para este proyecto: tan solo permite una publicación por cada 10 minutos.
  - Premium: La licencia premium exige un pago elevado, de 30000€ si bien es cierto que se realiza una única vez. (lifetime license)
  - Annual: Se proporciona acceso premium durante un año, sin embargo el coste es muy elevado subiendo a 15000€.

  Por tanto, debido al excesivo coste de las versiones de pago y a la gran limitación en tiempos de publicación para la versión gratuíta, el servicio Wolkabout debe ser descartado.

- Google Cloud Platform: Google, al igual que IBM proporcionan un ecosistema integral para el desarrollo de aplicaciones online, Iot, inteligencia artifical, etc… Facilitando la interconexión entre ellas.

Por tanto se vuelven opciones muy interesantes para el desarrollo del proyecto al permitir integrar el servicio IoT con el lado de aplicación.

Google proporciona un servicio administrado integral para el desarrollo IoT a través de su Cloud IoT Core reduciendo costes de inversión inicial.

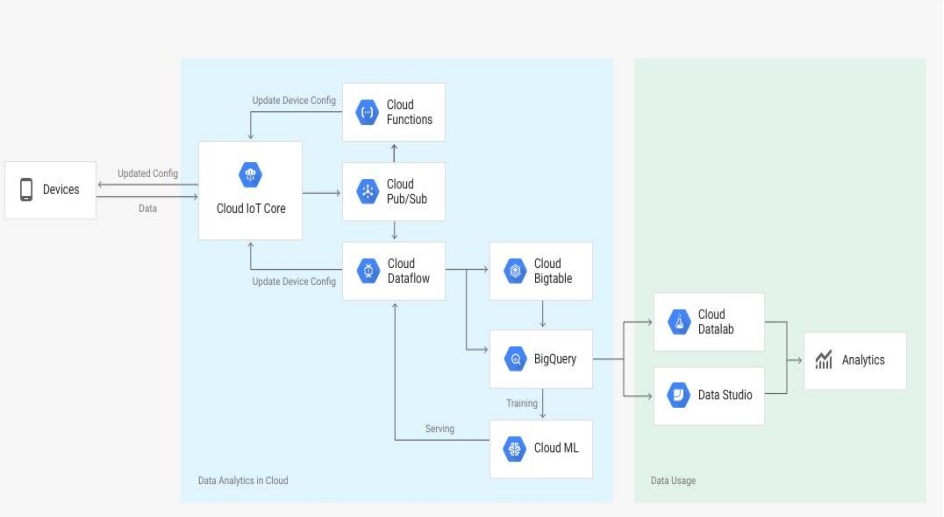


Ilustración 28:: Plataforma IoT de Google Cloud y analítica

A efectos de facturación Google toma como referencia los mensajes MQTT (connect, publish, etc…) y permite en su versión gratuita la publicación de datos hasta 250 MB mensuales, con atractivos precios por MB en tramos superiores.

Google Cloud permite también el desarrollo del lado de aplicación en sus propios servicios Cloud.

- IBMCloud IoT: Servicio integral al igual que ocurre con Google Cloud. En este caso el servicio IoT es integrable con la inteligencia artificial desarrollada por IBM Watson.

  En su versión gratuita, IBM permite la publicación de hasta 200 MB por mes.

Por otro lado permite la integración con aplicaciones desarrolladas en el propio servicio Cloud Foundry mediante sdk para multitud de lenguajes, entre los que se encuentra Python empleado en este proyecto.

Proporciona seguridad mediante TLS v1.2 y autenticación por token.

Para el desarrollo de este proyecto se han considerado las alternativas de Google e IBM, resultando finalmente elegida IBM Cloud.

4 Descripción técnica del sistema

A lo largo de las dos próximas secciones se realizará exposición sobre el sistema implementado desde desde dos perspectivas. Por un lado se explicará de forma detallada en la sección 5.1 el hardware empleado y en la sección 5.2 se expondrá el sistema desde la perspectiva de software.

En dichas exposiciones se seguirá un orden jerárquico partiendo de la base del sistema, es decir, desde el propio dispositivo terminal DMP, ascendiendo por el stack de subsistemas hasta alcanzar el servidor de almacenamiento cloud y el lado de aplicación.

4.1 Perspectiva Hardware

A continuación y en primer lugar se realizará la descripción del hardware seleccionado para la implementación del dispositivo DMP. Siguiendo la metodología expuesta en el capítulo primero, se comentarán los resultados de las primeras pruebas de uso de cada componente del hardware y su comparativa con lo esperado según sus hojas de datos.

Tal y como se ha explicado en capítulos anteriores, se ha seleccionado el dispositivo Hexiwear de Mikroelectronika para prototipado rápido.

Para la realización de este trabajo se han adquirido dos dispositivos Hexiwear con los accesorios de sujeción.

Si bien ya se han comentados en detalle las características de este dispositivo en el capítulo correspondiente, a la hora de realizar una primera prueba de programación con Zerynth Studio se requiere el uso de una de las dos estaciones de trabajo disponibles para Hexiwear.

Los requerimientos de este proyecto requieren la adición de dos funcionalidades además de las que proporciona el propio Hexiwear, por un lado se requiere conectividad GSM/GPRS y además se necesita geoposicionamiento GPS. Para ello se requiere el uso de dos clicks de expansión del catálogo de Mikroelectronika que implementen tales funciones.

Por ello, se ha elegido para este proyecto la estación de trabajo más simple denominada docking-station. Esta estación de trabajo incorpora tres conectores de MikroBus suficientes para conectar los dos clicks de expansión requeridos además de Hexiwear, pudiendo programar éste vía MicroUSB.

Por otro lado, la elección resulta obligada, ya que Zerynth solo permite virtualizar esta estación de trabajo.

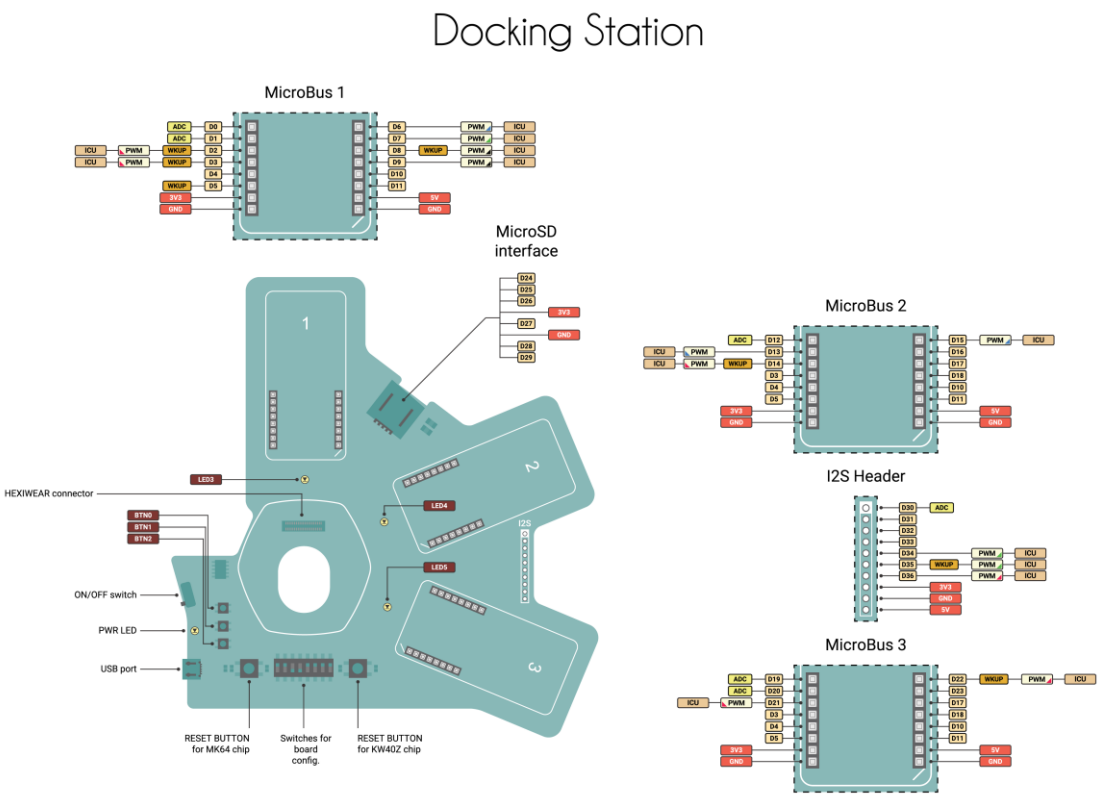


Ilustración 29:. Virtualización de dock-station

Para realizar la primera prueba de programación se inserta un dispositivo Hexiwear en el conector central, y se conecta la dock-station a través del puerto MicroUSB.

Para poder proceder al borrado de la memoria Flash y al grabado del nuevo código es preciso registrar el dispositivo en Zerynth a través del

menú correspondiente en el IDE. A continuación debe virtualizarse la placa, también empleando el menú del IDE. Tras estas dos operaciones el IDE ya reconoce el dispositivo Hexiwear conectado a dock-station y ya es su programación.

A través de los menús del IDE se puede visualizar la virtualización de la dock-station y de Hexiwear, pudiendo observar la distribución y asignación de pines(I/O, UARTs, ADC, etc…) a fin de darles el debido uso en el desarrollo de la aplicación.

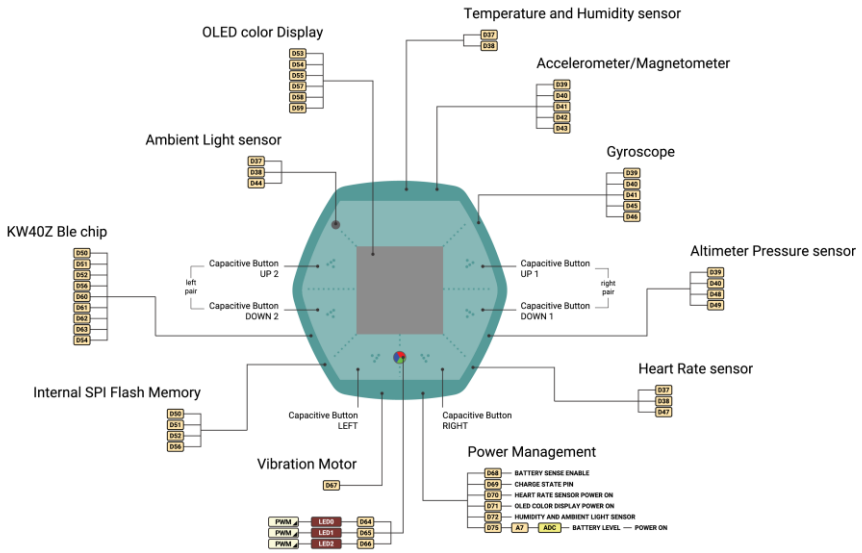


Ilustración 30:Virtualización de Hexiwear

.

Una vez registrada, virtualizada y conectada la dock-station a Zerynth Studio, para realizar la primera prueba se programa el dispositivo con el código de ejemplo para Hexiwear proporcionado por el propio IDE. Una véz ordenada la grabación, mediante el menú, se debe resetear el dispositivo mediante el botón de reset de la docking-station que actúa sobre el Kinetis MK64.

Una vez realizada la primera prueba con la dock-station, se procede a la elección de los clicks de expansión requeridos.

Para dotar al dispositivo de la funcionalidad de conectividad GSM/GPRS se elige el dispositivo de Ublox SARA G350, ya que satisface las

necesidades requeridas por el proyecto, no solo en cuanto a comunicación móvil sino a otras cuestiones de gran importancia como la seguridad en comunicaciones.

Este punto resulta de gran importancia ya que tal y como se ha expuesto en el capítulo explicativo de tecnologías disponibles en el mercado, el servicio cloud que se pretende emplear es el proporcionado por IBMCloud para IoT. Tal y como se ha comentado en su sección, este dispositivo emplea identificación para la conexión mediante token y requiere el uso del protocolo de seguridad TLS v1.2.

El dispositivo SARA G350 permite el uso de TLS en varias versiones incluída la citada para IBMCloud, lo que unido a otras especificaciones de interés como los tiempos de conexión y bajo coste le convierte en una buena elección.

La empresa Mikroelectronica ha desarrollado un click de expansión para este dispositivo G350 y lo denomina click GSM4.

Ilustración 31:Click Mikroelectronika GSM4

A continuación se detallan las características de este SARA G350:

- Comunicaciones cuatribanda 2.5G GSM/opGPRS
- Soporte de protocolos: TCP/IP, UDP, FTP, PPP, HTTP, SMTP
- Gobernado mediante comandos AT vía UART
- Máxima potencia de salida -8dBm
- Soporte para IPv4/IPv6

Su implementación en el click GSM4 añade las siguientes características:

- Voltaje de alimentación seleccionable de 3,3 V o 5 V mediante jumper onboard.
- UART accesible mediante microUSB onboard
- LEDs informativos
- Puerto de conexión SMA de antena GSM.
- Conexión de audio mediante puerto Jack para audio de llamada.
- Capacidad de recepción/envío de llamadas de audio/SMS
- SLOT para microSIM
- Conector MikroBus
- Obtención de voltajes de referencia mediante LDOs onboad.

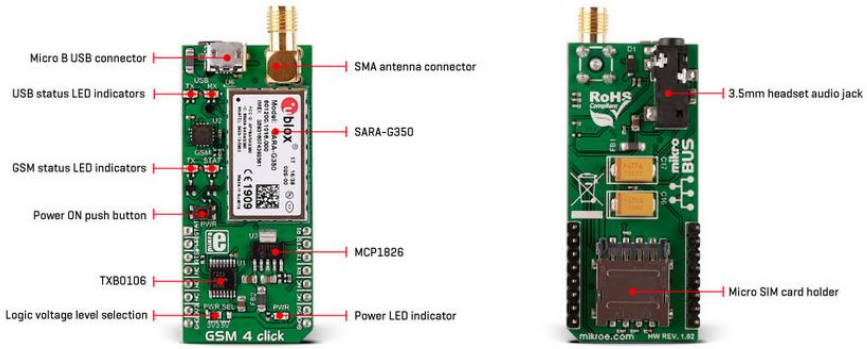


Ilustración 32:Componentes sobre Click GSM4

Debe considerarse el conexionado entre el dispositivo SARA G350 y los pines del conector MicroBus mostrado en la siguiente figura.

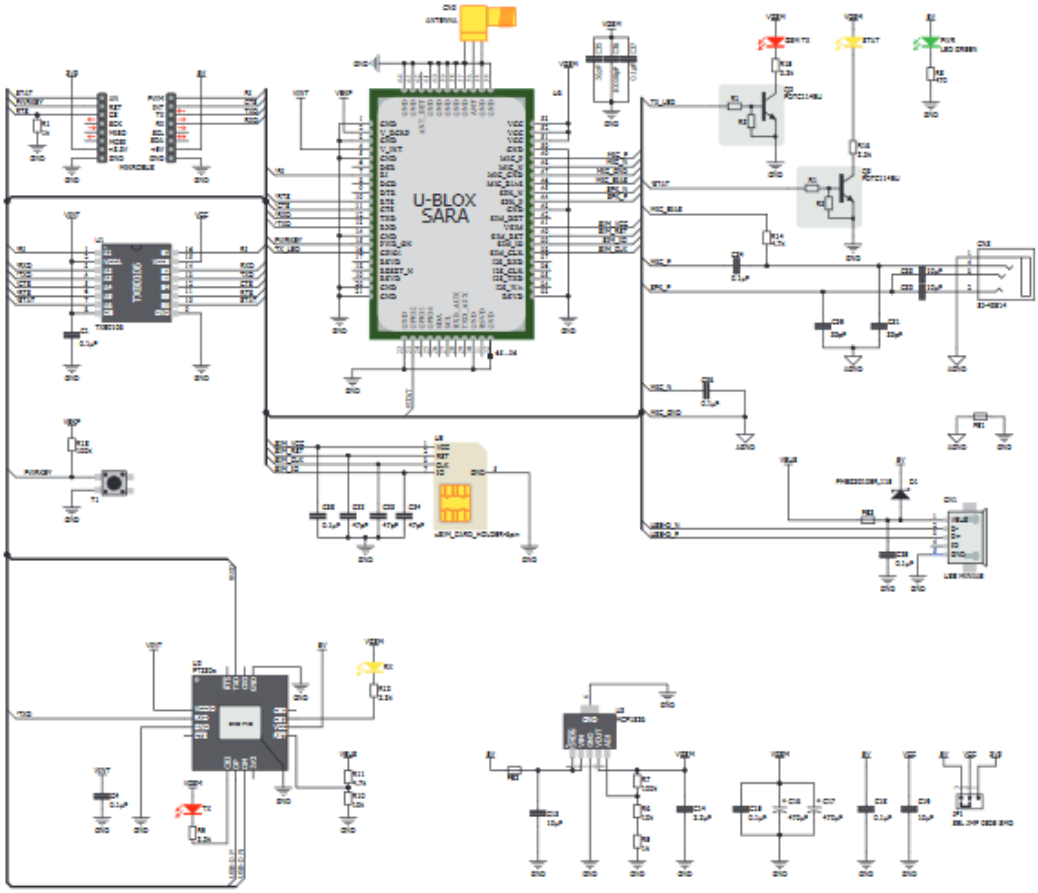


Ilustración 33: Esquema conexiones click GSM4

A la hora de realizar la primera prueba de programación, se emplea el código de ejemplo proporcionado por Zerynth, no obstante la librería proporcionada está diseñada para la gestión del propio SARA G350 y no el click que la contiene.

A fecha de realización de este proyecto, el click GSM4 no figura en la lista de Zerynth como compatible. Esto se observa al inspeccionar el código de ejemplo para la librería, en el que se incluye como parámetro para la función de inicialización del G350 el pin RTS para control de flujo por hardware.

No obstante inspeccionando la virtualización de la dock-station y el esquemático del click GSM4 se inicializa el dispositivo con los parámetros adecuados prescindiendo del RTS.

Para el proyecto el click se conecta sobre el conector MikroBus 1. Por tanto la comunicación de Hexiwear con el G350 se realizará a través de la UART etiquetada como SERIAL2 en la virtualización de la dock-station.

Se ha incluido una tarjeta SIM registrada con el operador Yoigo para logar la conectividad.

Una vez programado el dispositivo con el código de ejemplo, teniendo en cuenta lo referido anteriormente a su implementación en click de expansión, se logra establecer conexión con la red y recuperar información de la misma a través de los métodos de la clase GSM de Zerynth. Por tanto se concluye con éxito la primera prueba de programación o test de este click y se procede a la elección del correspondiente para la implementación de la funcionalidad GPS.

Zerynth no proporciona librerías para módulos GPS ni clicks de expansión que lo contengan. No obstante la comunicación con estos dispositivos se realiza via UART considerando adecuadamente las conexiones de Hexiwear con un eventual click GPS, puede implementarse la funcionalidad de geoposicionamiento sin requerir librería específica.

Para incorporar esta funcionalidad al proyecto se ha seleccionado el click de expansión NanoHornet que incorpora el módulo GPS NanoHornet org1411 del fabricante OriginGPS.

Ilustración 34:Click de expansión NanoGPS

El módulo NanoHornet org1411 incorpora antena GPS de tipo patch y ocupa un área de aproximadamente un centímetro cuadrado. Por lo que respecta a sus características técnicas

- Comunicación con MCU: UART, SPI, I2d
- Frecuencia: 1575.42 MHz
- Canales: 48
- Sensibilidad: -163 dBm
- Precisión 2.5m en el mejor caso.
- Tiempo de adquisición (Time to First Fix o TTFF):
    - Hot start <1 s en media
    - Warm start < 32 s en media
    - Cold start < 35 s media
    - Readquisición <1 s en media
- Consumo en adquisición: 11mW

Cabe destacar que los tiempos máximos para los diferentes arranque(en Hot, Warm, Cold) son calculados en media, por tanto cabe esperar que en ocasiones se vean incrementados en las pruebas de test para el proyecto.

Para la primera prueba de programación, el click de expansión se conecta al segundo conector MikroBus de la dock-station.

Por tanto la comunicación de Hexiwear con el org1411 se efectuara a través de la UART denominada SERIAL3 en la virtualización de la dock-station.

Dado que no existe librería para el dispositivo en Zerynth tampoco existe código de ejemplo para realizar la primera prueba de programación. Por ello se ha realizado un pequeño código de test que inicializa el dispositivo y adquiere la longitud y latitud además de la altitud, además

de mostrar por pantalla del monitor serie el vector completo de respuesta devuelto por el módulo GPS tras la petición de adquisición.

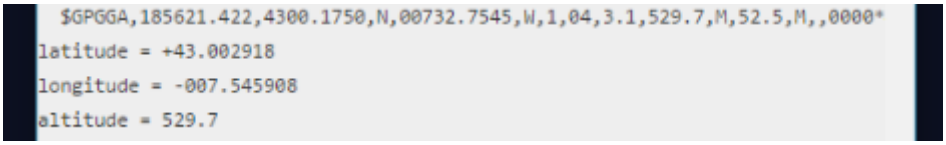


Ilustración 35: Datos obtenidos por el código de test

Se han realizado diversas pruebas en entorno urbano y rural a fin de verificar la precisión y los tiempos de adquisición vistos en las hojas de datos y de su comparativa se extraen los siguientes puntos:

- Como límite inferior de precisión se han observado grandes fallos de precisión cuando se enlazan un numero no superior de tres satélites. A partir de 4 satélites se ha observado mejora en la precisión desde un caso más desfavorable de 32 m de error hasta un error de 5m para 8 satélites.

  Cabe señalar, que la precisión indicada en el datasheet del org1411, inferior a 2.5 m, no esta garantizada por el fabricante en el 100% de casos ya que según indican las propias hojas de datos, esta precisión se ha obtenido en el 50% de los tests.

- El TTFF desde frio cumple ha cumplido lo especificado en las hojas de datos (página 15 del manual org1411) alcanzando en todos los casos 35 s.

- Se ha verificado el tiempo de readquisición de 1 segundo.

- En los diversos test realizando Hot start se ha verificado el TTFF de 1 s. Por lo que respecta a distancia de error, se prueba con 4 satélites (información completa) y se obtiene un error de 32 m,. Se repite con 8 y se reduce a 12,5 m.

Por tanto el dispositivo Hexiwear con la configuración de clicks de expansión conectados a través de la dock-station queda como se muestra en la siguiente figura:

Ilustración 36:Montaje completo mostrando el logo DMP

Una vez observados los componentes de hardware seleccionados para expandir las funcionalidades de Hexiwear, es clara la falta de portabilidad del dispositivo completo para pruebas al incluir la dock-station para permitir la conexión de los clicks de expansión.

Además, en caso de desconectar el dispositivo del laptop vía MicroUSB, la batería de Hexiwear perecería en un corto lapso de tiempo, debido a los consumos energéticos especialmente altos de los clicks de expansión, debido a la naturaleza de los mismos.

A efectos de realización de pruebas se ha adquirido un docking extra para Hexiwear. Este nuevo dispositivo permite conectart Hexiwear con un único click de expansión, e incorporar baterías extra en forma de 4 pilas AAA.

Si bien su uso exige sacrificar una funcionalidad, en este caso el GPS, permite testear el comportamiento del dispositivo manteniendo las comunicaciones con el servidor cloud, permitiendo además por sus dimensiones, el portarlo a modo de pulsera.

Ilustración 37:Dispositivo extensor de batería

Finalmente, por lo que respecta al resto de la arquitectura del proyecto, no se emplea otro tipo de hardware al emplearse servicios cloud propietarios.

Para la implementación del lado de aplicación del NCG, se emplea un Notebook Jupyter en Python 3 corriendo localmente en un laptop conectado a la red.

4.2 Perspectiva Software

Concluida la exposición de anteriores capítulos, se ha definido ya que tecnologías de software se emplearán para orquestar la arquitectura de la solución al proyecto.

En términos de herramientas de desarrollo, lenguajes de programación y plataformas de servicios de software la arquitectura se articula del modo que se muestra en la figura siguiente:

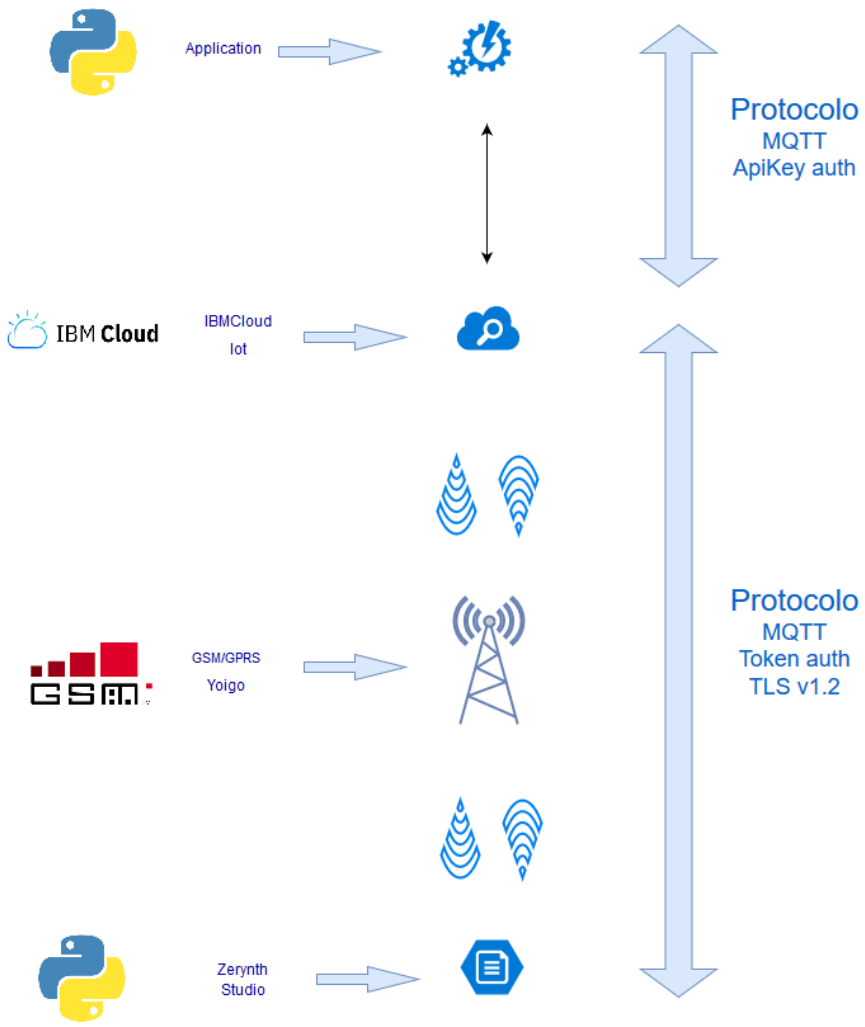


Ilustración 38:Arquitectura de tecnologías software

Para realizar una descripción de la arquitectura de software mostrada en la figura anterior se procederá explicando desde la cima de la pila, es decir, desde el lado de aplicación, para descender hasta la descripción del dispositivo embedded DMP.

Por lo que respecta al lado de aplicación, una vez establecido que el servicio cloud a emplear en el proyecto sería el proporcionado por IBM,

parece lógico emplear los propios servicios de computación en la nube de IBM para el desarrollo de la misma.

En efecto, la aplicación se puede desarrollar, almacenar y ejecutar en IBM Cloud Foundry, desarrollando la aplicación en el lenguaje deseado que en este caso sería Python. Por tanto en esta opción se podría desarrollar la aplicación mediante Python y desplegarla en IBM Cloud mediante Flask.

Sin embargo, dado que el objetivo principal de este proyecto no es tanto el desarrollar el lado de aplicación, como el desarrollar una arquitectura básica, es decir, un primer prototipo funcional que permita realizar una fase de testing completa, la aplicación a desarrollar es relativamente simple.

Se pretende una aplicación capaz de suscribirse a las publicaciones de todos los DMPs conectados a IBMCloud, capaz de enviar ciertos comandos a cada DMP, y finalmente publicar información de configuración para cada DMP, lo que implica que estos habrán de estar suscritos a las mismas.

Por tanto se implementarán callbacks en ambos lados de la arquitectura, tanto en dispositivo DMP, como en aplicación.

Como conclusión a esta relativa simplicidad en la aplicación requerida, y a efectos de realización de este proyecto, se ha optado por una solución intermedia.

Por un lado se han generado las credenciales de aplicación en los servicios IBMCloud para obtener la ApiKey correspondiente que permita enlazar una aplicación directamente con el servicio IoT generado.

Por otro lado se ha optado por desarrollar la aplicación en local, es decir, su desarrollo y ejecución se realizará en un ordenador portátil, accediendo al servicio IoT mediante las credenciales generadas anteriormente.

La aplicación se ha confeccionado mediante un NoteBook de Jupyter, lo que permite que este autodocumentada mediante lenguaje MarkDown,

facilitando así la lectura del código considerando que se trata de un proyecto de ámbito académico.

Ilustración 39:Jupyter NoteBook App del NCG

La aplicación, procesa dos eventos. Por un lado el evento de inicialización de los dispositivos DMP. Este evento es generado por IBMCloud de forma automática cuando un dispositivo se conecta con éxito al servico IoT. El evento de conexión se genera por el mero hecho de realizar la conexión aunque el dispositivo no publica nada.

La aplicación se suscribe a la generación de los eventos de conexión que son procesados por la función de callback definida en la misma.

IBMCloud proporciona a la función de callback de la aplicación información suficiente para la identificación unívoca del dispositivo conectado.

Como respuesta a cada conexión, la aplicación envía información de configuración en formato JSON al dispositivo recién conectado.

De este modo se centraliza la configuración de cada dispositivo en el lado de aplicación sin ser necesario acceder y manipular presencialmente cada dispositivo que se pretenda reconfigurar.

La información de configuración particulariza el funcionamiento del dispositivo en cada faceta:

- Permite establecer el período de muestreo de información de sensores particularizado para cada sensor en concreto, así como desactivarlos.
- Permite conmutar el uso del módulo GPS para posicionamiento. De este modo el posicionamiento puede pretenderse mediante adquisición GPS siendo este el modo más preciso o mediante información de la conexión a la red GSM y sus parámetros como la RSSI. Esta diferenciación permite desde el nivel de aplicación establecer patrones de uso del GPS en función de características como lugar de despliegue del dispositivo: rural frente a urbano; o para establecimiento de políticas de ahorro energético.
- No se considera la desconexión GSM del dispositivo, éste deberá estar conectado a la nube en todo momento.

En el otro extremo de la arquitectura de la solución, se encuentran los dispositivos DMP.

Para el desarrollo de este proyecto se encuentran las siguientes librerías proporcionadas por Zerynth:

- Streams: Permite comunicación serie
- JSON: Gestión de mensajes JSON. Utilizado para carga de credenciales de conexión IBMCloud y publicación de mensajes en el servicio IoT.
- NXP.hexiwear: incorpora todos los métodos necesarios para la gestión del dispositivo hexiwear: sensores, pulsadores, OLED…
- Queue: Permite la gestión de colas de mensajes entre threads en ejecución. En este proyecto, se emplea una cola para enviarle al módulo de gestión de pantalla OLED, aquellos mensajes que se deseen publicar.

- Ublox.g350: gestiona el módulo SARA G350 para comunicación GSM. Se emplea para inicializar el módulo.
- Wireless.gsm: La librería GSM se emplea sobre la conectividad proporcionada por el módulo anterior. La clase GSM permite conectar a un APN de la red GSM, recuperar información de la red, del propio punto de acceso, rssi entre otros.
- Ibmcloud.iot: Permite realizar la conexión mediante token y TLS al servicio IoT de IBM. Una vez realizada la conexión, mediante la clase MQTT se procede a la conexión de protocolo, publicación, bucle de espera para callbacks y desconexión eventual del servicio.
- Timers: permite la gestión de temporizadores en Zerynth.

Tras el arranque del dispositivo conectado a alimentación, éste lleva a cabo un proceso de inicialización que consiste en:

- Muestra el logo DMP en pantalla durante inicialización.
- Carga de credenciales IBMCloud desde archivo JSON.
- Inicialización de la clase Hexiwear.
- Inicialización del módulo g350
- Conexión a IBMCloud mediante MQTT y token.

El dispositivo estructura la información mostrada por pantalla estableciendo la línea superior de la misma con un ancho de 10 píxels a modo de barra de estado.

Sobre esta línea presentará información sobre conectividad: esquina superior izquierda imprimirá “GSM” cuando exista conexión a la red; sobre la esquina superior derecha imprimirá “CLOUD” si la conexión MQTT es lograda.

Ilustración 40:Barra estado DMP

Cada paso en la inicialización expresada en líneas superiores será acompañado de un mensaje en la pantalla principal, a estos efectos, el hilo principal de ejecución enviará una notificación vía cola de mensajes (librería queue) al módulo de control de pantalla OLED desarrollado en el proyecto

Tal y como se ha comentado en el capítulo que analizaba opciones posibles como proveedores de servicio IoT cloud, no ha sido posible lograr la conexión con IBMCloud en un primer intento.

Esto es debido a la exigencia de niveles de seguridad proporcionados por TLS v1.2 por parte de IBM. Si bien esta opción de seguridad se puede desconectar a través del panel de control de IBMCloud, no resulta interesante para el proyecto, ya que sí que se pretende implementar niveles de seguridad en la transmisión de los datos debida la naturaleza personal de los mismos.

Zerynth proporciona sus librerías en código abierto lo que permite realizar modificaciones y personalizaciones necesarias para el proyecto.

Tal y como se había indicado en la sección anterior, analizando el hardware empleado, el módulo Ublox SARA G350 soporta la versión de TLS que requiere IBMCloud con lo que a priori no debería haber mayor dificultad para resolver este problema.

En efecto, inspeccionando la librería para SARA G350 escrita en lenguaje C, (g350.c), se observa que en la línea 2389 donde se configura el uso del protocolo TLS, se indica por defecto una versión inferior a la requerida por IBMCloud.

Para resolver el problema se procede a modificar esta línea para obligar al módulo a emplear TLS v1.2. La librería queda finalmente como se muestra:

```
2388    //set minimum version TLS1.0 Changed to support TLS 1.2(IBM cloud)
2389    //if(_gs_tls_config(1,1,NULL,0)) goto exit;
2390    if(_gs_tls_config(1,3,NULL,0)) goto exit;
2391
```

Ilustración 41:Modificación realizada a la librería g350.c

Tras esta modificación el sistema se conecta correctamente a IBMCloud y comienza a publicar.

Una vez conectado, IBMCloud genera el evento de conexión y el lado aplicación responde enviando mensaje de configuración al DMP recién conectado.

Durante el proceso de inicialización se lanzan dos hilos de ejecución paralelos al hilo base o inicial.

Por un lado mediante el método Threads() se lanza la función main_thread() del módulo screen.py desarrollado en el proyecto.

Este hilo será el encargado de procesar la cola de mensajes a través de la que recibirá indicaciones para imprimir notificaciones a través de la pantalla OLED.

Por otro lado, se genera otro hilo paralelo mediante el método Loop() de la clase MQTT que permite realizar la escucha de comandos y mensajes que pueda enviar el servicio IoT o el lado aplicación para lanzar la ejecución de la función callback correspondiente.

En el hilo principal se procesa la captura de datos de los enlaces y la publicación de los datos en el servicio cloud.

Además se adquieren las coordenadas de posición GPS si la configuración del dispositivo así lo ordena.

Por lo que respecta al procesado de datos para la generación de alarmas ante situaciones de peligro o riesgo para el usuario, este se dejará a la parte aplicación montada en el servicio IoT.

Si bien no se desarrolla en este proyecto, este desarrollo permitirá desplegar procesado de datos de gran potencia sobre los generados aplicando tecnologías de machine-learning que permitan la mejor extracción de features posible de los datasets generados.

Se plantea la posibilidad de emplear técnicas de machine learning basadas en redes neuronales para detectar posibles situaciones de riesgo para la salud del usuario en función de los datos obtenidos, mediante modelado del comportamiento humano.

Se plantea la utilización de redes neuronales recurrentes o RNN sobre los datos proporcionados por la tríada acelerómetro, giroscopio, magnetómetro, para detectar posibles caídas del usuario o situaciones de indisposición.

Por lo que respecta a procesado de datos de modo local en el propio dispositivo DMP, se procesarán datos de ciertos sensores generando alarmas locales que alerten o aconsejen comportamientos al usuario.

Particularmente se procesarán los datos adquiridos del sensor de temperatura, de este modo ante exposición detectada de temperaturas extremas, bien bajas o bien altas, se emitirá mensaje de alerta al usuario a través de la pantalla OLED y del motor de vibración.

Analizando el DMP desde un punto de vista de funcionamiento, tras el arranque, y una vez realizada la conexión con la red GSM, y con el servicio cloud de IBM, se muestran los respectivos indicadores en la barra de estado superior de la pantalla mostrada en la ilustración 39. A continuación se realiza la carga de la configuración pregrabada en el dispositivo en el archivo *sensor.conf.json.* Los parámetros de configuración se listan a continuación:

- "tglobal": "5000": Establece un período de muestreo de sensores de 5 segundos
- "ttemp": "on": Indica que se tomarán datos del sensor de temperatura
- "thumid":"on", Indica que se tomarán datos del sensor de humedad
- "tpress": "on", Indica que se tomarán datos del sensor de presión
- "tlum": "on", Indica que se tomarán datos del sensor de intensidad lumínica
- "mimu_items" : "20", Establece 20 medidas de sensores MIMU por muestreo según tglobal.
- "mimu_period" : "200", Establece un período de 200 ms entre cada una de las mimu_items medidas.
- "GPS" : "off", prefija el GPS desactivado.

A continuación el dispositivo muestra en su parte inferior una barra de estado de la batería, de color verde, amarillo o un mensaje en rojo en función de su nivel.

Una vez que el DMP se conecta a la nube de IBM, la aplicación desarrollada en Python que simula el NCG detecta el evento *connect* y mediante una función *callback* para dicho evento, envía en un comando información de configuración al DMP recién conectado, con el mismo

formato que la pregrabada en el dispositivo. De este modo desde el NCG se puede configurar el DMP.

A partir de este momento el DMP se sitúa en un bucle en el que muestrea sus sensores y elabora un diccionario de resultados para enviar a la nube.

El modo de conformar dicho diccionario depende de la configuración establecida, bien la prefijada, o bien la descargada del NCG.

Mientras el dispositivo permanece en dicho modo de funcionamiento, comprueba además la temperatura exterior a la que se expone el usuario. En caso de que esta se sitúe por debajo de una mínima preestablecida o por encima de otra máxima también prefijada, el DMP muestra en la parte central de su pantalla un icono informativo(copo de nieve o sol radiante) y un mensaje de advertencia a modo de aviso al usuario.

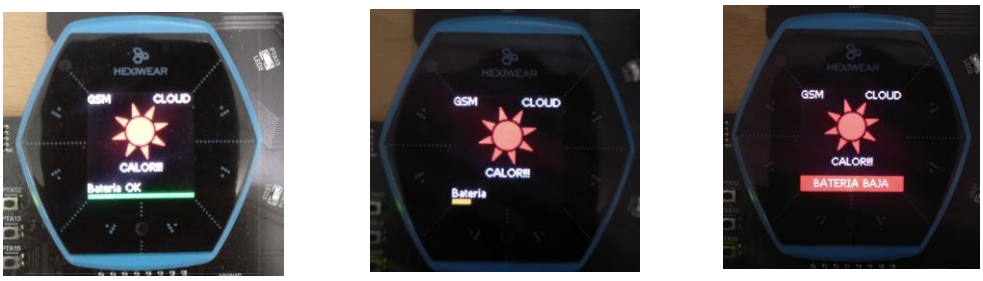


Ilustración 42: Mensaje de calor excesivo y estado de batería

Durante este tiempo, el NCG puede enviar comandos al DMP a través de sus credenciales vía protocolo MQTT. Tal y como se ha comentado anteriormente no se ha implementado el frontend de usuario, sin embargo en el código del NCG_SIM, se incluyen tres celdas que permiten emular esta funcionalidad, enviando mensajes al DMP de tres tipos:

- Cita: Emula la citación al usuario por parte de un médico a consulta. En el mensaje se incluye la fecha y hora. En el momento de la recepción el DMP muestra al usuario un mensaje en la parte central de la pantalla con la información enviada y el tema cita.

- Medicina: Se reserva este mensaje para el aviso de forma automática por parte del NCG de recordatorios para la toma de medicación. Al acercarse la hora de la toma de una medicación el NCG enviará un mensaje al DMP con el nombre de la medicina y mostrará en la parte central de la pantalla el mensaje “Tomar” acompañado del nombre del producto.
- Actividad: Si en NCG detectase poca actividad física o sedentarismo excesivo por parte del usuario, el NCG enviaría un aviso indicando el nivel de actividad detectado, y al igual que en los demás casos se mostraría en la parte central de la pantalla.

Analizando el DMP desde nivel de programación, el proyecto consta de los siguientes módulos:

Nombre	Descripción
Main.py	Módulo principal, realiza la conexión GSM y Cloud, además de contener el Loop principal de adquisición de datos de sensores.
Screen.py	Gestiona la pantalla de Hexiwear, recibe los mensajes a través de una cola. En ella se recibe el nombre del método a ejecutar dentro de screen.py y parámetros necesarios. Mediante la traducción a través de un diccionario se ejecuta el método correspondiente al mensaje o imagen a mostrar en la pantalla.
Org1411.py	Funciones para gestión el GPS nanoHornet.
InitDMP.py	Inicializa el DMP, muestra el logo creado para el dispositivo y contiene la función de adquisición de datos de los sensores y construcción del diccionario de información.
Icons.py	Contiene tres iconos: Sol radiante, copo de nieve, caja de herramientas para los sucesos de alarma por temperatura extrema alta/baja y para

	casos de fallo de comunicaciones respectivamente.
LogoDMP.py	Contiene el logo creado para el DMP
Helpers.py	Contiene dos funciones auxiliares para la carga de los datos de configuración pregrabada y credenciales de dispositivo para IBMCloud
Device.conf.json	Almacena las credenciales de dispositivo para acceso a IBMCloud
Sensor.conf.json	Almacena la configuración pregrabada del DMP

Tabla 3: Módulos Python del DMP

Por lo que respecta al emulador del NCG, tal y como se ha mencionado anteriormente este consta de dos bloques. Uno permite emular el lado aplicación, es decir el propio NCG y otro el lado cliente, emulando DMP para pruebas.

5 Trabajo futuro

A partir de la prueba de concepto desarrollada a través de este proyecto, se abren fundamentalmente tres líneas de trabajos futuros:

- Continuación del desarrollo del dispositivo embebido DMP. Mediante el desarrollo de este proyecto se ha obtenido uno de los productos, un dispositivo con funcionalidades mediante hardware requeridas por los objetivos establecidos en el capítulo primero. Ahora bien, se trata de una prueba de concepto que sirve como punto de partida hacia un producto comercial.

  Por tanto resta, a fin de recorrer dicho camino, un amplio abanico de tareas a realizar.

  Desde la perspectiva de software cabe considerar tareas de securización, análisis de consumo de energía, control exhaustivo de errores, y ampliación de funcionalidades.

  Desde la perspectiva de hardware, es preciso desarrollar una nueva placa base para el DMP en la que se contengan los dispositivos onboard de Hexiwear y además se integren los módulos externos utilizados en el prototopo, es decir, el Ublox G.350 y el GPS nanoHornet Origin 1411.

- Implementación de tecnologías de análisis de datos masivos. Esta prueba de concepto permite la publicación en un servicio Cloud, en este caso IBMClout para IoT, de los datos recolectados por el conjunto total de DMPs. Por tanto, una línea de trabajo futuro se sitúa sobre la implementación de soluciones de Big Data, para un eficaz análisis de estos datos. Esta implementación puede realizarse directamente sobre el servicio cloud mediante el servicio CloudFoundry, que permite alojar aplicaciones

desarrolladas en diversos lenguajes de programación, con acceso a los datos almacenados en IBMIoT mediante API Key.

- Desarrollo del FrontEnd cliente: Tal como se comentó en primeros capítulos, el frontend con diversos clientes: servicio médico, farmacia, servicio asistencial, familiares; no se ha implementado, simulando algunas transacciones de comandos de forma *hardcoded* en el aplicativo del NCG_SIM. Por tanto el desarrollo de este frontend constituye una línea más de trabajo a desarrollar.

- Integración de agente conversacional (IA): Finalmente, cabe señalar la posibilidad de integrar un sistema de inteligencia artificial sobre la arquitectura propuesta.

  Otros proyectos cuyo ámbito de acción coinciden con el del presente, tales como Activage, emplean tal y como se observó en la sección 2.3, una central telefónica con operadores empleados para dar asistencia telefónica a los usuarios del sistema.

  Se propone como línea de trabajo futuro, la implementación de un agente conversacional o chatbot que permita liberar carga de trabajo de estos centros estableciendo un lazo cerrado de comunicación con el usuario a través del propio dispositivo DMP.

  Empleando el módulo UBlox G.350, se puede lanzar llamada telefónica desde el NCG hacia el usuario, actuando como interlocutor un sistema de inteligencia artificial que permita confirmar conclusiones extraídas del análisis de datos, y seguimiento de la situación del usuario de forma diaria.

  Para el desarrollo de este nuevo sistema, IBM proporciona su tecnología conversacional Watson, con capacidad en idioma castellano y adaptado a conversación telefónica. Se trata de un servicio de pago, tarificado por llamada API y por tiempo de

llamada. Si bien los costes son asumibles. Por otra parte es necesario emplear una tecnología que permita el lanzamiento y la gestión de la propia llamada(grabación, interacción, etc…)

Una tecnología disponible en el mercado para este objetivo es Twilio que mediante SIP trunk puede enlazarse con la plataforma IBMCloud y de este modo realizar una orquestación de los servicios requeridos.

Se ha realizado una prueba de funcionamiento de este sistema AI propuesto. Si bien no se ha desarrollado el cuerpo del agente conversacional se ha conseguido lanzar llamada telefónica y obtener respuesta básica inicial de Watson como interlocutor. Para ello se han empleado los siguientes servicios de IBMCloud:

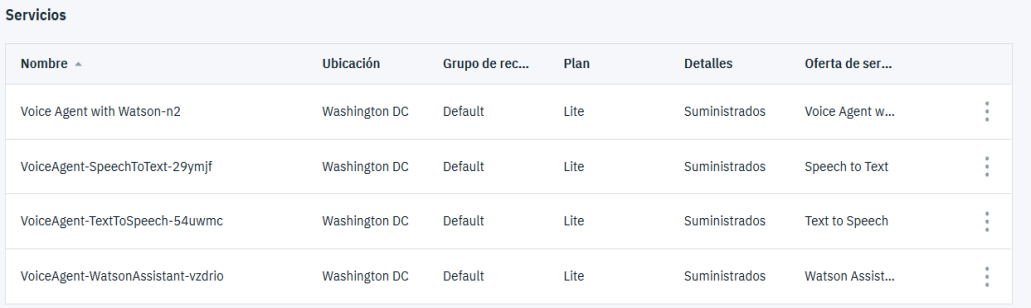


Ilustración 43: Servicios IBM para implementar Chatbot

Además se ha creado una cuenta en el proveedor Twilio con un número telefónico español para el NCG enlazado con Voice Agent with Watson de IBM mediante SIP Trunk:

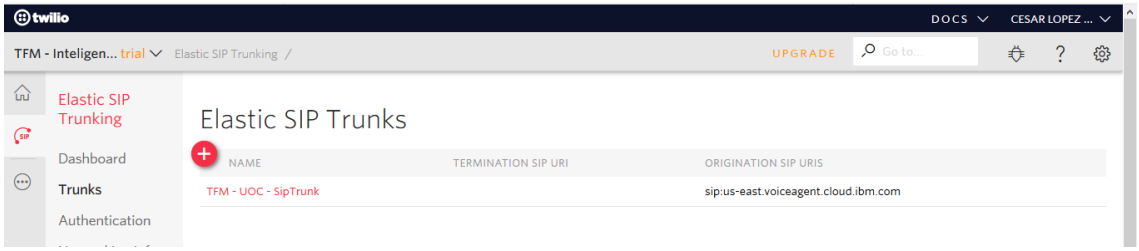


Ilustración 44:Creación de SIP TRUNK en Twilio

6 Estudio financiero

Se elabora la siguiente tabla de costes en el desarrollo de prototipo:

Elemento	Descripción	Coste
Hexiwear	Dos dispositivos Hexiwear	98€
GSM 4	Click expansión GSM/GPRS con Ublox G350	49 €
Click NanoHornet	Click con módulo GPS Origin 1411	45 €
Antena GSM	Antena GSM para Ublox G350	6.9 €
Extensor baterías	Placa para Hexiwear con capacidad para baterías y un click de expansión	25 €
Dock-Station	Estación de programación para Hexiwear	39€
Servicios IBMCloud	Coste cero para desarrollo hasta 200MB de transmisión	0
		Total de costes: 262 €

Tabla 4: Bill of Materials (BOM)

7 Conclusiones

Se han conseguido los objetivos establecidos. Se obtiene un prototipo a modo de prueba de concepto basado en tecnologías válidas para la continuación del desarrollo del sistema hasta un producto final comercial.

Se ha obtenido un gran bagaje de conocimientos, habiendo estudiado un amplio abanico de tecnologías implicadas en el conjunto total de la arquitectura end-to-end.

Se ha analizado una alternativa de desarrollo embedded en alto nivel basada en Python, que se revela como una herramienta válida para el desarrollo, permitiendo acortar sensiblemente los plazos de desarrollo, todo ello, claro está en detrimento de la eficiencia.

Se han analizado también tecnologías de servicio en la nube, eligiendo finalmente IBMCloud. Esto ha requerido un análisis de la gran cantidad de servicios que permite esta plataforma, desde almacenamiento de datos IoT, hasta la interconexión de lado aplicación y dispositivos vía protocolo MQTT. Además se han analizado otros servicios del ámbito de la inteligencia artificial como herramientas para líneas futuras de trabajo, pudiendo desarrollar el conjunto total del sistema dentro del mismo proveedor IBM.

Además desde el lado aplicación se ha logrado implementar la conectividad con el DMP a través del propio servicio cloud realizando una prueba de concepto de extremo a extremo.

8 Glosario

DMP: Dispositivo monitor de personas

NCG: Nodo central general

GPS: Global positioning system

IDE: Entorno para desarrollo de proyectos de software

CallBack: Funcionalidad de protocolo MQTT para recepción de comandos

UART: Dispositivo para transmisión y recepción serie de modo asíncrono.

MQTT: Protocolo de comunicaciones para IoT

IoT: Internet of Things

Thread: Hilos de ejecución en entorno multitarea en tiempo real

Cola: Implementación FIFO para comunicación de hilos de ejecución

JSON: Formato de texto para transmisión de datos

9 Bibliografía

[CES] Dispositivo monitor de personas dependientes

César López Bermúdez

2017, TFG UOC

http://openaccess.uoc.edu/webapps/o2/bitstream/10609/59452/7/clopezbermTFG0117mem%C3%B2ria.pdf

[RTO] FreeRTOS

Web, comunidad y manual del sistema operativo en tiempo real FreeRTOS para sistemas embebidos.

Fecha: 10/01/2019

https://www.freertos.org/

[ZER] Zerynth Docs

Web de documentación sobre la plataforma Zerynth y comunidad. Fecha: 10/01/2019

https://docs.zerynth.com/latest/index.html

[ACT] Proyecto ACTIVAGE

Web, e información sobre el proyecto ACTIVAGE.

Fecha: 10/01/2019

www.activageproject.eu/

[PSA] Nisar Ahmad, Master Thesis in embedded computing system. Università di Pisa. 2015

[CLA] Claudia Villalonga Palliser, Ontology engineering and reasoning to support real world human behavior recognition. Universidad de Granada. 2016

[BAN] E. Monton, J.F. Hernandez, J.M. Blasco, T. Herve, J. Micallef, I. Grech, A. Brincat and V. Traver. Body area network for wireless patient monitoring. 2008

10 Anexos

Implementación del DMP:

Módulo main.py

```
# DMP creado por César López Bermúdez
# Fecha 15/10/2018

import streams
import json
from nxp.hexiwear import hexiwear
import InitDMP as dmp
import queue
import screen
import org1411 as gps

from ublox.g350 import g350
from wireless import gsm
#import requests
#import json

from ibmcloud.iot import iot
import helpers
import timers

####### Código principal

## Inicialización comm serie
streams.serial()
sleep(1000)




fifo_screen = queue.Queue(maxsize=20)
new_resource('device.conf.json')
new_resource('sensor.conf.json')

device_conf = helpers.load_device_conf()

device = iot.Device(device_conf['device_id'], device_conf['device_type'],
device_conf['organization'], device_conf['auth_token'])


## Inicialización de dispositivo Hexiwear
```

```
print("Creando objeto Hexiwear")

hexi = hexiwear.HEXIWEAR(battery_en=True, oled_en=True,
amb_light_en=True, heart_rate_en=False,
                         temp_humid_en=True, gyro_en=True,
acc_magn_en=True, pressure_en=True,
                         bt_driver_en=False)


def gps_connect():
    try:

        # init gps module
        drv_gps = SERIAL3
        pwr_gps = D15
        wup_gps = D12

        print("init gps")


        gps.init(drv_gps, pwr_gps, wup_gps)
        sleep(3000)
        print("listo gps")
        for i in range(20):
            # retrieve global position
            print("- - - - - - - - - Intento ",i)
            res = gps.get_pos()
            if res != 0:
                lat, lon, alt = res
                print("latitud = %+10.6f" % lat)
                print("longitud = %+011.6f" % lon)
                print("altitud= %.1f" % alt)
                sleep(5000)
            sleep(1000)

    except Exception as e:
        print(e)
    return res

def GSM_Connect():

    """

.. py:function:: GSM_Connect()

    Realiza la conexión GSM a la red via G350
    Retorna código (OK = 0, ERR = -1)
    """
```

```
    ret_err = g350.init(SERIAL2,D19,D2,D0,D20)

    sleep(1000)

    for i in range(20):
        try:
        # set here the APN name
            gsm.attach('Internet')
            break
        except g350Exception:
            print("lksdf")
        except TimeoutError:
            print("lj")
        sleep(2000)
    else:
        print("fds")
        while True:
            sleep(1000)

    return 0

def IBMCloud_Connect():
    try:
        print('connecting to mqtt broker...')
        device.mqtt.connect()
    except Exception as err:
        return -1
    return 0

# Funciones para Callback del NCG
# Carga de configuración

def config_call(msg):
    global sens_config
    sens_config = msg

def medic_call(med):
    global med_msg
    med_msg = med

def cita_call(cita):
    global cita_msg
    cita_msg = cita

def activ_call(activ):
    global activ_msg
    activ_msg = activ
```

```
# Carga de configuración predeterminada
sens_config = helpers.sensor_conf()
med_msg = {}
cita_msg = {}
activ_msg = {}

dmp.init_settings(hexi)
sleep(1000)

thread(screen.main_thread,fifo_screen,hexi)


### fin coments

# Lanzar Icono GSM
fifo_screen.put(['GSM_Con','0'])

for _ in range(10):
    return_connect = GSM_Connect()
    if return_connect == 0:
        break
else:
    fifo_screen.put(['Tech_Fail','0'])

# Retira mensaje GSM
fifo_screen.put(['GSM_Con','0'])

# Lanza GSM_UP en barra superior de estado

fifo_screen.put(['GSM_UP','0'])

## Realización de la conexión de arranque inicial.
## Se exige que esta conexión se realice, en caso
## Contrario se supone avería y se demanda servicio
## Técnico

# Tomar hora y fecha de Internet


### Conexión con IBMCloud IoT
### Envío del mensaje de arranque

fifo_screen.put(['MQTT_Con','0'])

fifo_screen.put(['Tech_Fail','0']) if  IBMCloud_Connect() == -1 else
fifo_screen.put(['CLOUD_UP','0'])
sleep(2000)
```

```
fifo_screen.put(['MQTT_Con','0'])

#Referencia de funciones para calkbacks
device.on_cmd('init', config_call)
device.on_cmd('medi', medic_call)
device.on_cmd('cita', cita_call)
device.on_cmd('acti', activ_call)


device.mqtt.loop()
print("inicializado Loop MQTT")

while 1:
    # Loop principal
    dict_sensors = dmp.get_sensors(hexi,sens_config)
    sleep(500)

    # comprobar callbacks y mostrar datos en pantalla Hexiwear
    if cita_msg:
        fifo_screen.put(['cita',cita_msg['Medicina']])
        cita_msg = {}
    if med_msg:
        fifo_screen.put(['med',med_msg['fecha']])
        med_msg = {}
    if activ_msg:
        fifo_screen.put(['activ',activ_msg['Nivel']])
        activ_msg = {}


    # Comprobar posibles alarmas locales
    print("Loop Main\n\n",sens_config) #, dict_sensors)
    sleep(500)

    if dict_sensors['temperature'] <= 2:
        print("FRIIIIO\n")
        fifo_screen.put(['Temp_Low','0'])
        hexi.vibration(200)
        sleep(300)
        fifo_screen.put(['Temp_Low','0'])

    if dict_sensors['temperature'] > 32:
        print("CALORRRR\n")
        fifo_screen.put(['Temp_High','0'])
        hexi.vibration(200)
        sleep(300)
        fifo_screen.put(['Temp_High','0'])

    # adición de información de GSM al diccionario
    #RSSI
```

```
    dict_sensors['GSM_rssi'] = gsm.rssi()
    net_info = gsm.network_info()
    # Guarda la estación base
    dict_sensors['GSM_node'] = net_info[3]

    # Control de Batería
    fifo_screen.put(['Bateria', str(dict_sensors['bat_level'])])

    # Comprobar si config exige gps
    if sens_config["GPS"] == "on":
        total = gps_connect()
        latit, long, alti = total
        dict_sensors['GPS_lat'] = latit
        dict_sensors['GPS_long'] = long


    #device.publish('main', json.dumps(dict_sensors))

    #Establecimiento del período de muestreo
    sleep(int(sens_config['tglobal']))
```

Módulo InitDMP.py

```
"""
.. module:: InitDMP

*******************************
InitDMP: módulo de inicializción
*******************************

Este módulo inicializa dispositivos y ajusta settings iniciales.

Consta de las siguientes funciones:

"""


import LogoDMP as DMP
import timers


## Inicialización comm serie


def init_settings(hexi):
    """

.. py:function:: init_settings(hexi)

    Inicializa el hardware ajustando los settings iniciales tras
arrancado y presenta logotipos en pantalla OLED de la UOC además del
propio del DMP

    :param hexi(*hexiwear*): Objeto Hexiwear para acceso a sus métodos

    """
    # Muestra logo UOC

 #   hexi.draw_image(UOC.LogoUoc, 32, 32, 32, 32)

    #hexi.vibration(100)
    sleep(3000)
    # Muestra logo DMP
    print("DMP")
    hexi.draw_image(DMP.LogoDMP, 8, 8, 80, 80)
    sleep(3000)
    hexi.clear_display()
    sleep(1000)
    return
```

```
def pressed_up(hexi):
    """

.. py:function:: pressed_up(hexi)

    Detecta pulsación del botón UP

    :param hexi(*hexiwear*): Objeto Hexiwear para acceso a sus métodos

    """
    hexi.vibration(100)


def pressed_down(hexi):
    """

.. py:function:: pressed_down(hexi)

    Detecta pulsación del botón DOWN

    :param hexi(*hexiwear*): Objeto Hexiwear para acceso a sus métodos

    """
    hexi.vibration(100)


def toggle_ble(hexi):
    """

.. py:function:: toggle_ble(hexi)

    Toggle de la capacidad BLE (Kinetis)

    :param hexi(*hexiwear*): Objeto Hexiwear para acceso a sus métodos

    """
    try:
        print("Left Button Pressed")
        hexi.vibration(100)
        hexi.bt_driver.toggle_adv_mode()
    except Exception as e:
        print("error on left_pressed", e)


def get_sensors(hexi, sens_conf):
    """

.. py:function:: get_sensors(hexi)
```

```
    Adquiere datos de los sensores onboard

    :param get_sensors(*hexiwear*): Objeto Hexiwear para acceso a sus
métodos

    """



    bl, chg = hexi.get_battery_level(chg_state=True)
    al = hexi.get_ambient_light() if sens_conf['tlum'] == "on" else False
    temp = hexi.get_temperature() if sens_conf['ttemp'] == "on" else
False
    humid = hexi.get_humidity() if sens_conf['thumid'] == "on" else False
    press = hexi.get_pressure() if sens_conf['tpress'] == "on" else False

    dict_sens = {'bat_level': bl, 'bat_stat' : chg, 'light_ambient' : al,
'temperature' : temp,'humidity' : humid, 'pressure' : press }
    mimu = {}
    # Captura secuencia de valores de Acc, Giro, Magnetometer
    # F = 1 KHz
    for i in range(int(sens_conf['mimu_items'])):
        acc = hexi.get_accelerometer_data()
        magn = hexi.get_magnetometer_data()
        giro = hexi.get_gyroscope_data()
        temp = timers.now()
        mimu[i] = {'acc' : acc, 'magn' : magn, 'giro' : giro, 'time' :
temp}
        sleep(int(sens_conf['mimu_period']))

    #Integra datos en un diccionario
    dict_sens['mimu'] = mimu

    return dict_sens
```

Módulo Screen.py

```
import queue
import icons



def show_msg(hexi,*args):
    """
      lines= 0 indica status bar
    """
    line_msg=55
    for ar in args:
        hexi.draw_text(ar,0, line_msg, 96, 15, align=3, color=0xFFFF,
background=0x0000, encode=False)
        line_msg-=20
    return


def cita(hexi,msg):
    show_msg(hexi,msg,"Tomar")

def activ(hexi,msg):
    show_msg(hexi,msg,"Actividad")

def med(hexi,msg):
    show_msg(hexi,msg,"Medico")

def GSM_Con(hexi,*args):
    show_msg(hexi,"Red GSM","Conectando")


def GSM_UP(hexi,*args):
    hexi.draw_text("GSM",0, 0,10,10, align=3, color=0xFFFF,
background=0x0000, encode=False)


def CLOUD_UP(hexi,*args):
    hexi.draw_text("CLOUD",60, 0,90,10, align=3, color=0xFFFF,
background=0x0000, encode=False)

def Time_Now(hexi,*args):
    show_msg(hexi,"httpbin.org","Request hora")

def Bateria(hexi, nivel):
    largo_barra = int(nivel)-4
```

```
    if largo_barra < 15:
        hexi.draw_text("BATERIA BAJA",0, 80,96,15, align=3, color=0xFFFF,
background=0xF800, encode=False)
    else:
        hexi.draw_text("Bateria",0, 80,96,11, align=1, color=0xFFFF,
background=0x0000, encode=False)
        hexi.fill_rect(0,92,96,4,0x0000, encode=False)
        #sleep(200)
        hexi.fill_rect(0,92,largo_barra,4,0xDEC0, encode=False) if
largo_barra < 60 else hexi.fill_rect(0,92,largo_barra,4,0x07E0,
encode=False)

def MQTT_Con(hexi,*args):
    show_msg(hexi,"IBMCloud IoT","Conectando..")


def Tech_Fail(hexi,*args):
    hexi.clear_display()
    show_msg(hexi,"AVERIA!!!")
    hexi.draw_image(icons.Tools, 22, 10, 50, 50)
    sleep(10000)

def Temp_Low(hexi,*args):
    hexi.fill_rect(0,11,96,70,0x0000, encode=False)
    hexi.draw_text("FRIO!!!",0, 55, 96, 26, align=3, color=0xFFFF,
background=0x0000, encode=False)
    #show_msg(hexi,"Frio!!!")
    hexi.draw_image(icons.Snow, 22, 10, 50, 50)

def Temp_High(hexi,*args):
    hexi.fill_rect(0,11,96,70,0x0000, encode=False)
    hexi.draw_text("CALOR!!!",0, 55, 96, 26, align=3, color=0xFFFF,
background=0x0000, encode=False)
    hexi.draw_image(icons.SunHot, 22, 10, 50, 50)

def Clock(hexi,hora):
    hexi.draw_text(hora,0, 15, 96, 60, align=3, color=0xFFFF,
background=0x0000, encode=False)

def main_thread(fifo_screen,hexi):
    on_screen = []

    method_dict = {
        'GSM_Con': GSM_Con,
        'GSM_UP': GSM_UP,
        'CLOUD_UP': CLOUD_UP,
        'Time_Now': Time_Now,
        'MQTT_Con': MQTT_Con,
        'Tech_Fail': Tech_Fail,
```

```
        'Temp_Low': Temp_Low,
        'Temp_High': Temp_High,
        'Clock': Clock,
        'cita' : cita,
        'med' : med,
        'activ': activ,
        'Bateria': Bateria

    }

    while True:
        try:
            next_screen = fifo_screen.get()
            on_screen.append(next_screen) if not next_screen in on_screen
else  on_screen.remove(next_screen)
        except Exception as e:
            print(e)
        for exe_method in on_screen:
            method_dict[exe_method[0]](hexi,exe_method[1])
            if exe_method[0] =='Bateria' or exe_method[0] =='cita' or
exe_method[0] =='med' or exe_method[0] =='activ':
                on_screen.remove(exe_method)
        sleep(2000)
```

Módulo helpers.py

```
import json


def load_device_conf():
    confstream = open('resource://device.conf.json')
    conf = ''
    while True:
        line = confstream.readline()
        if not line:
            break
        conf += line
    return json.loads(conf)

def sensor_conf():
    confstream = open('resource://sensor.conf.json')
    conf = ''
    while True:
        line = confstream.readline()
        if not line:
            break
        conf += line
    return json.loads(conf)
```

Módulo org1411.py

```
import streams
import timers

new_exception(org1411Exception, Exception)

_ser = None
_pwr = None
_wup = None


def _send(cmd):
    _ser.write(cmd + '\r\n')


def _read(timeout=2000):
    t = timers.timer()
    t.start()
    while not _ser.available():
        if t.get() > timeout:
            return "timeout"
    return _ser.readline().strip('\r\n').strip(chr(0))

# send messages to stop continous stream
def _set_freq():
    _send('$PSRF103,02,00,00,01*26')
    _send('$PSRF103,03,00,00,01*27')
    _send('$PSRF103,04,00,00,01*20')
    _send('$PSRF103,00,00,00,01*24')


def get_pos():
    #send request for gga message
    _send('$PSRF103,00,01,00,01*25')
    res = _read()
    print("Vector respuesta en gps.get_pos[35]:\n ",res)
    return _parse_gga_lla(res)


def _parse_gga_lla(res):
    parts = res.split(',')
    # msg_id      = parts[0]
    # utc_time    = parts[1]
    latitude    = parts[2]
    latitude_d  = parts[3]
    longitude   = parts[4]
    longitude_d = parts[5]
    # pos_fix     = parts[6]
```

```
    # n_satell    = parts[7]
    # hdop        = parts[8]
    altitude    = parts[9]
    # alt_units   = parts[10]
    # geoid_sep   = parts[11]
    # geoid_units = parts[12]
    # age_diff_c  = parts[13]
    # diff_ref    = parts[14]

    if len(latitude) > 4:
        lat = int(latitude[:2]) + float(latitude[2:]) / 60
        if latitude_d == 'S':
            lat = -lat

        lon = int(longitude[:3]) + float(longitude[3:]) / 60
        if longitude_d == 'W':
            lon = -lon

        alt = float(altitude)

        return lat, lon, alt
    else:
        return 0


def _pulse():
    digitalWrite(_pwr, HIGH)
    sleep(100)
    digitalWrite(_pwr, LOW)


def init(ser_drv, pwr_pin, wup_pin, rst_pin=None):
    global _pwr, _wup

    _pwr = pwr_pin
    _wup = wup_pin

    if _ser is not None:
        _ser.close()
        _ser = None

    _ser = streams.serial(ser_drv, set_default=False, baud=4800)

    pinMode(_wup, INPUT)
    pinMode(_pwr, OUTPUT)

    _pulse()
    sleep(50)
    while not digitalRead(_wup):
```

```
        _pulse()
        sleep(50)

    t = timers.timer()
    t.start()
    while _read() != '$PSRF150,1*3E':
        if t.get() > 2500:
            raise org1411Exception

    _set_freq()
```

Módulo device.conf.json

```
{
    "organization": "XXXXXXX",
    "device_id": "101",
    "device_type": "RightWristDev",
    "auth_token": "XXXXXXXXXXXXXXXX"
}
```

Módulo sensor.conf.json

```
{
    "tglobal": "5000",
    "ttemp": "on",
    "thumid":"on",
    "tpress": "on",
    "tlum": "on",
    "mimu_items" : "20",
    "mimu_period" : "200",
    "GPS" : "off"
}
```

Implementación del NCG en Python

```
# coding: utf-8

# <div class=pull-right>
# Trabajo de fin de Máster</div/>
# Máster Universitario de Ingeniería de Telecomunicaciones

# ![Logo de la
UOC](https://upload.wikimedia.org/wikipedia/commons/a/a3/Logo_blau_uoc.pn
g)

# **Alumno: César López Bermúdez**
#
# ** Trabajo de fin de Máster Ingeniería de telecomunicaciones**
#
# ** Fecha: 10/01/2018 **
#
# ** Area de electrónica **
#

# <h1>Table of Contents<span class="tocSkip"></span></h1>
# <div class="toc"><ul class="toc-item"><li><span><a href="#Código-para-
simulación-de-NCG" data-toc-modified-id="Código-para-simulación-de-NCG-
1"><span class="toc-item-num">1  </span>Código para simulación
de NCG</a></span><ul class="toc-item"><li><span><a href="#Carga-de-
librerías" data-toc-modified-id="Carga-de-librerías-1.1"><span
class="toc-item-num">1.1  </span>Carga de
librerías</a></span></li><li><span><a href="#Definición-de-funciones-
para-gestión-de-Callbacks" data-toc-modified-id="Definición-de-funciones-
para-gestión-de-Callbacks-1.2"><span class="toc-item-
num">1.2  </span>Definición de funciones para gestión de
Callbacks</a></span></li><li><span><a href="#Carga-de-credenciales-
ApiKey-y-auth_token" data-toc-modified-id="Carga-de-credenciales-ApiKey-
y-auth_token-1.3"><span class="toc-item-num">1.3  </span>Carga
de credenciales ApiKey y auth_token</a></span></li><li><span><a
href="#Suscripción-a-publicaciones-DMPs" data-toc-modified-
id="Suscripción-a-publicaciones-DMPs-1.4"><span class="toc-item-
num">1.4  </span>Suscripción a publicaciones
DMPs</a></span></li><li><span><a href="#Envío-de-notificaciones-a-DMP-
vía-MQTT" data-toc-modified-id="Envío-de-notificaciones-a-DMP-vía-MQTT-
1.5"><span class="toc-item-num">1.5  </span>Envío de
notificaciones a DMP vía MQTT</a></span><ul class="toc-item"><li><span><a
href="#Envío-de-Aviso-de-toma-de-medicinas" data-toc-modified-id="Envío-
de-Aviso-de-toma-de-medicinas-1.5.1"><span class="toc-item-
num">1.5.1  </span>Envío de Aviso de toma de
medicinas</a></span></li><li><span><a href="#Envío-de-Cita-con-médico"
```

```
data-toc-modified-id="Envío-de-Cita-con-médico-1.5.2"><span class="toc-
item-num">1.5.2  </span>Envío de Cita con
médico</a></span></li><li><span><a href="#Envío-de-actividad-física-baja"
data-toc-modified-id="Envío-de-actividad-física-baja-1.5.3"><span
class="toc-item-num">1.5.3  </span>Envío de actividad física
baja</a></span></li></ul></li><li><span><a href="#Desconexión-del-
servicio-NCG" data-toc-modified-id="Desconexión-del-servicio-NCG-
1.6"><span class="toc-item-num">1.6  </span>Desconexión del
servicio NCG</a></span></li></ul></li><li><span><a href="#Código-para-
simular-cliente-de-IoT" data-toc-modified-id="Código-para-simular-
cliente-de-IoT-2"><span class="toc-item-num">2  </span>Código
para simular cliente de IoT</a></span><ul class="toc-item"><li><span><a
href="#Credenciales-para-simular-dispositivo" data-toc-modified-
id="Credenciales-para-simular-dispositivo-2.1"><span class="toc-item-
num">2.1  </span>Credenciales para simular
dispositivo</a></span></li><li><span><a href="#Inicializa-dispositivo-y-
realizar-la-conexión" data-toc-modified-id="Inicializa-dispositivo-y-
realizar-la-conexión-2.2"><span class="toc-item-
num">2.2  </span>Inicializa dispositivo y realizar la
conexión</a></span></li><li><span><a href="#Definición-de-funciones-de-
Callback-para-comandos-recibidos" data-toc-modified-id="Definición-de-
funciones-de-Callback-para-comandos-recibidos-2.3"><span class="toc-item-
num">2.3  </span>Definición de funciones de Callback para
comandos recibidos</a></span></li><li><span><a href="#Realización-de-la-
conexión" data-toc-modified-id="Realización-de-la-conexión-2.4"><span
class="toc-item-num">2.4  </span>Realización de la
conexión</a></span></li><li><span><a href="#Desconexión-de-cliente" data-
toc-modified-id="Desconexión-de-cliente-2.5"><span class="toc-item-
num">2.5  </span>Desconexión de
cliente</a></span></li></ul></li></ul></div>

# # Código para simulación de NCG

# ## Carga de librerías

# In[ ]:


import ibmiotf as ibm
import time
import sys
import pprint
import uuid
import ibmiotf.application
import ibmiotf.device


# ## Definición de funciones para gestión de Callbacks
```

```
# In[ ]:


# Función para definir el callback para Eventos

def myAppEventCallback(event):
    print("recibido %s (%s) %s: h=%s x=%s" % (event.deviceId,
event.deviceType, event.timestamp.strftime("%H:%M:%S"), data['hello'],
data['x']))
    appCli.disconnect()


# Función para definir el callback para connect
def myStatusCallback(status):
    global devtip
    global devid
    devtip = status.deviceType
    devid = status.deviceId
    print("RECIBIDO EL STATUS Y ES ",status.action," - DispoId ",
status.deviceId," - Tipo ",status.deviceType)
    if status.action == "Connect":
        time.sleep(5)
        cmdData={
            "tglobal": "5000",
            "ttemp": "on",
            "thumid":"on",
            "tpress": "on",
            "tlum": "on",
            "mimu_items" : "20",
            "mimu_period" : "200",
            "GPS" : "off"
        }
        appCli.publishCommand(status.deviceType, status.deviceId, "init",
"json",cmdData)


# ## Carga de credenciales ApiKey y auth_token

# In[ ]:


# Credenciales para aplicación

organization = "epyy4h"

appId = "NCG_SIM"
#authKey = "XXXXXXXXXX"
appMethod =  "apikey"
#apptoken = “XXXXXXXXXXXXX”
```

```
authKey = "XXXXXXXXXXXXXXXXXXX"
apptoken = “XXXXXXXXXX"

deviceType = "RightWristDev"
deviceId = "101"


# ## Suscripción a publicaciones DMPs

# In[ ]:




# Inicializar aplicación cliente
try:
    appOptions = {"org": organization, "id": appId, "auth-method":
appMethod, "auth-key" : authKey, "auth-token": apptoken, "clean_session"
: True}
    appCli = ibmiotf.application.Client(appOptions)
except Exception as e:
    print(str(e))
    sys.exit()


appCli.connect()

# Suscripción a status(para registro del connect y disconnect)
appCli.deviceStatusCallback = myStatusCallback
appCli.subscribeToDeviceStatus()


#appCli.subscribeToDeviceEvents(deviceType, deviceId, "Initialized")
#appCli.deviceEventCallback = myAppEventCallback


# ## Envío de notificaciones a DMP vía MQTT

# Se simula el envío de notificaciones al dispositivo DMP. Para ello se
presuponen conocido el identificador del dispositivo cliente. Este será
proporcionado por un agente externo a través de un FrontEnd determinado.
A modo de ejemplo se simulan envíos de notificaciones médicas, expedidas
por facultativos a través de un posible aplicativo. Mediante una base de
datos se hallaría el ClientId del DMP objetivo a partir de los datos del
paciente/usuario. Este paso se obvia, considerando el ClientId fijado al
definido para pruebas.

# ### Envío de Aviso de toma de medicinas
```

```
# In[ ]:


cmdData1={"Medicina" : "Adolonta"}
appCli.publishCommand(devtip, devid, "medi", "json",cmdData1)


# ### Envío de Cita con médico

# In[ ]:


cmdData2={"fecha" : "12/10 15:00"}
appCli.publishCommand(devtip, devid, "cita", "json",cmdData2)


# ### Envío de actividad física baja

# In[ ]:


cmdData3={"Nivel" : "bajo"}
appCli.publishCommand(devtip, devid, "acti", "json", cmdData3)


# ## Desconexión del servicio NCG

# In[ ]:


# Se realiza la desconexión
appCli.disconnect()


# # Código para simular cliente de IoT

# ## Credenciales para simular dispositivo

# In[ ]:


devauthToken = “XXXXXXXXXXXXX"
devauthMethod = "token"


# ## Inicializa dispositivo y realizar la conexión

# In[ ]:
```

```
# Initialize the device client.
try:
    deviceOptions = {"org": organization, "type": deviceType, "id":
deviceId, "auth-method": devauthMethod, "auth-token": devauthToken}
    deviceCli = ibmiotf.device.Client(deviceOptions)
except Exception as e:
    print("Caught exception connecting device: %s" % str(e))
    sys.exit()


# ## Definición de funciones de Callback para comandos recibidos

# In[ ]:


def cmdcallback(cmd):
    print("Comando: ",cmd.command,"\n","Contenido :",cmd.data)

deviceCli.commandCallback = cmdcallback


# ## Realización de la conexión

# In[ ]:


deviceCli.connect()


# ## Desconexión de cliente

# In[ ]:



#Disconnect the device and application from the cloud
deviceCli.disconnect()
```