ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA
GRADO EN INGENIERIA DE COMPUTADORES

# SISTEMA PARA LA MONITORIZACIÓN Y ASISTENCIA EN EL HOGAR USANDO OPENHAB Y ROBÓTICA MÓVIL

# MONITORING AND HOME ASSISTANCE USING OPENHAB AND MOBILE ROBOTICS

Realizado por
**Víctor del Sol Toledano**
Tutorizado por
**Cipriano Galindo Andrades**
**Francisco Ángel Moreno Dueñas**
Departamento
**Ingeniería de Sistemas y Automática**

UNIVERSIDAD DE MÁLAGA
MÁLAGA, NOVIEMBRE DE 2016

Fecha defensa:
El Secretario del Tribunal

# RESUMEN

Este trabajo está enfocado a la creación de un sistema informático, robótico y automático con el que podamos actuar en un entorno, por ejemplo, una casa, dependiendo de la lectura que realicemos de los diferentes dispositivos instalados y de la actividad actual de los usuarios del sistema.

El sistema actualiza en tiempo real el estado de los diferentes puntos y elementos del domicilio a través de los sensores instalados. El estado de los sensores es comunicado a un servidor que procesa los datos recibidos y los transmite a la plataforma *openHAB*, en la que, mediante diferentes algoritmos, se estudia la posible actividad que está realizando el usuario.

Una vez establecida la actividad que está realizando el usuario y las condiciones del entorno, se procede a realizar, en caso de necesidad, cualquier tipo de ayuda, aviso o proposición al usuario, una comunicación con el exterior o alguna modificación en el domicilio, mediante la comunicación con el actuador, en nuestro caso un robot móvil.

El sistema, a través de la plataforma *openHAB*, dispone de una interfaz gráfica accesible desde cualquier navegador.

Palabras claves: openHAB, robótica móvil, domótica, automatización, monitorización.

# ABSTRACT

This project is focused on the creation of a computerized, robotic and automatic system which allows us to operate in an environment, as a house, for example, according to the readings of a set of installed devices and the current activity of the system's users.

The system is updated with the information gathered from the installed sensors in real-time. The sensors' status is communicated to a server that processes the received data and transmits it to the openHAB platform, where some simple user's activities are inferred through a set of different algorithms.

Once the user's activity and the environment conditions have been established, an actor, in our case a mobile robotic unit, will be able to perform a particular set of actions: give support, advice or a proposition to the user, communicate something to the exterior, or a modification in the house.

The system has a graphic interface accessible from any browser, via the openHAB platform.

Keywords: *openHAB*, mobile robotic, domotic, automation, monitoring

# ÍNDICE

# Capítulo 1.

# INTRODUCCIÓN

La evolución de diferentes aspectos de la sociedad, así como de los avances tecnológicos y científicos en el campo de la medicina han mejorado enormemente las condiciones de vida y, con ello, ha aumentado la esperanza de vida.

Debido a esto, el envejecimiento de la población es algo real y que tenemos actualmente en la sociedad. Como podemos observar en la figura 1, en 2014, la sociedad de los países europeos tenía un porcentaje superior al 17% de media de población mayor de 65 años. Solo había un país que no tenía más de un 11% de población mayor de 65 años, Turquía (7%), mientras que el gran grueso de los países se encontraban entre el 16% y 18%, entre los que se encontraba España. Países como Italia y Alemania llegaban a cifras muy cercanas al 21%.

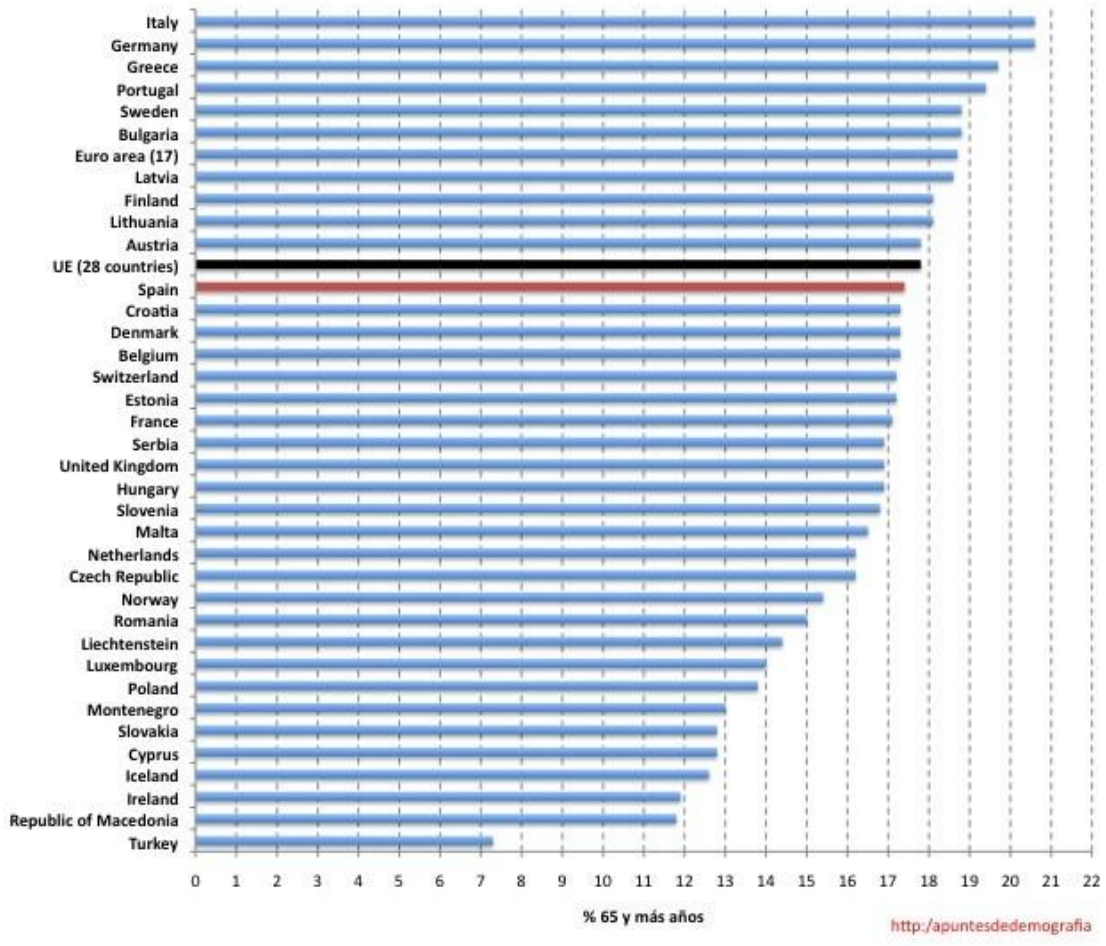



Figura 1. Distribución del porcentaje de población mayor de 65 años por países en 2014. Fuente: eurostats [1]

Actualmente, y como podemos observar en el mapa de la figura 2, la gran mayoría de países europeos tiene un porcentaje de población mayor de 65 años superior al 17.7%. Comparando ambas estadísticas podemos concluir que el porcentaje de población mayor de 65 años está aumentando en los países europeos de forma constante.

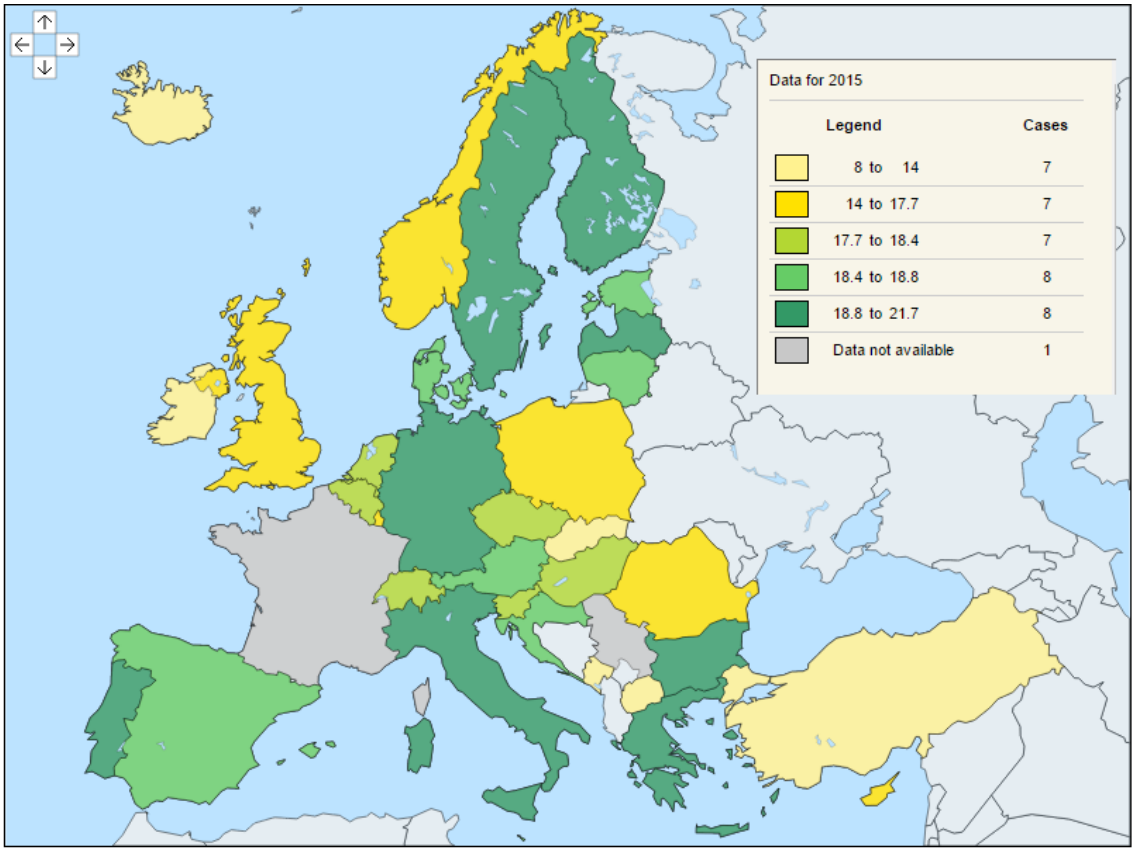



Figura 2. Mapa de distribución de porcentaje de población mayor de 65 años en Europa en 2016. Fuente: INE

El INE [2] (Instituto Nacional de Estadística) sitúa que en España hay entorno a un 18.5% de población mayor de 65 años y estima que este porcentaje se incrementará notablemente en el próximo medio siglo. Concretamente, se estima que en torno a 15 millones de personas de los más 41 millones de personas que conformarán la sociedad española sean mayores de 65 años, es decir, más de un 36% de la sociedad española será mayor de 65 años en el año 2050.

Respecto al estudio de la proyección de la población podemos ver, en la figura 3, que, en los colores más claros, referentes a la actualidad, los picos más amplios, tanto en varones como en mujeres, son a la edad de los 40 o 45 años y, a partir de ahí se va reduciendo el número de personas.

Si nos fijamos en cambio en los colores más oscuros, de nuevo tanto en hombres como en mujeres observamos que el mayor pico se produce en los 80 u 85 años, es

decir, que la estimación es que la franja de edad de 60 a 100 años sea la que tenga un mayor número de personas en la sociedad en un futuro próximo.

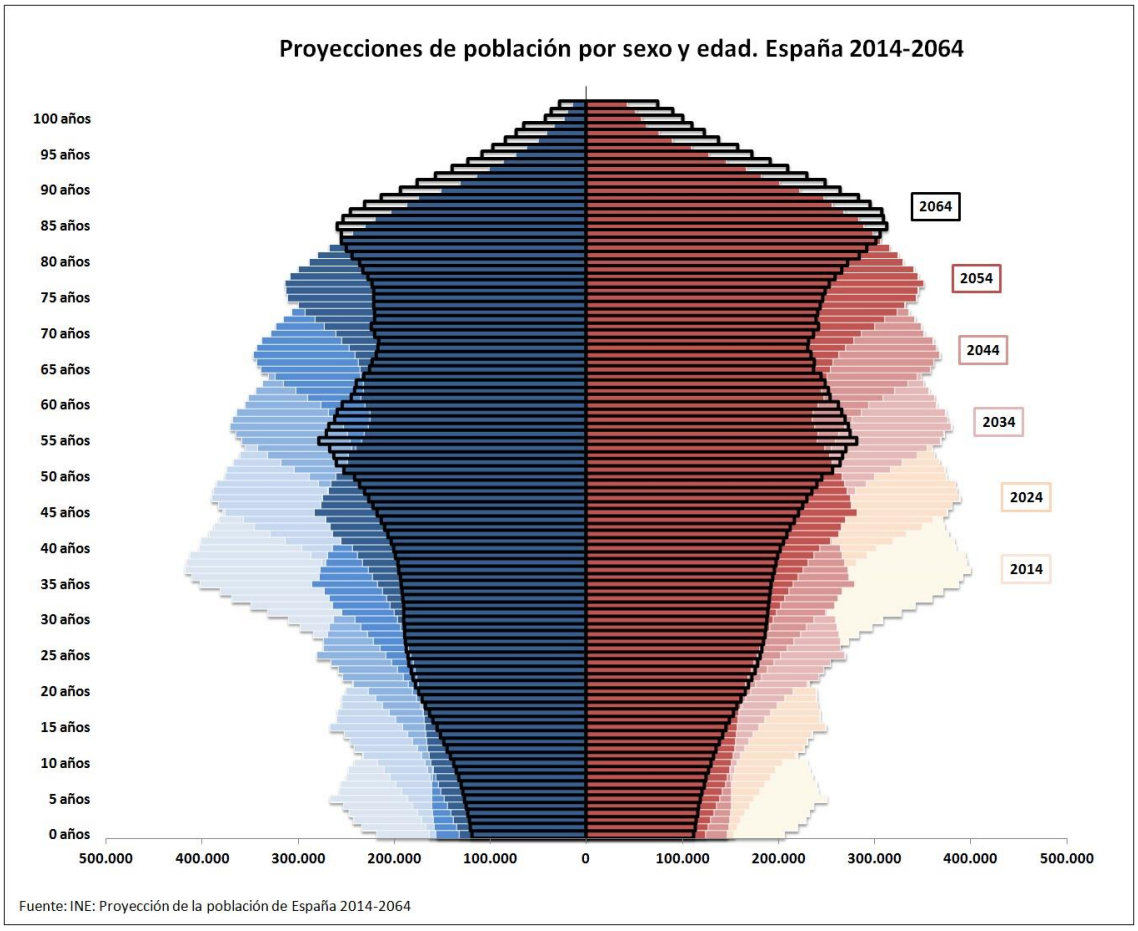
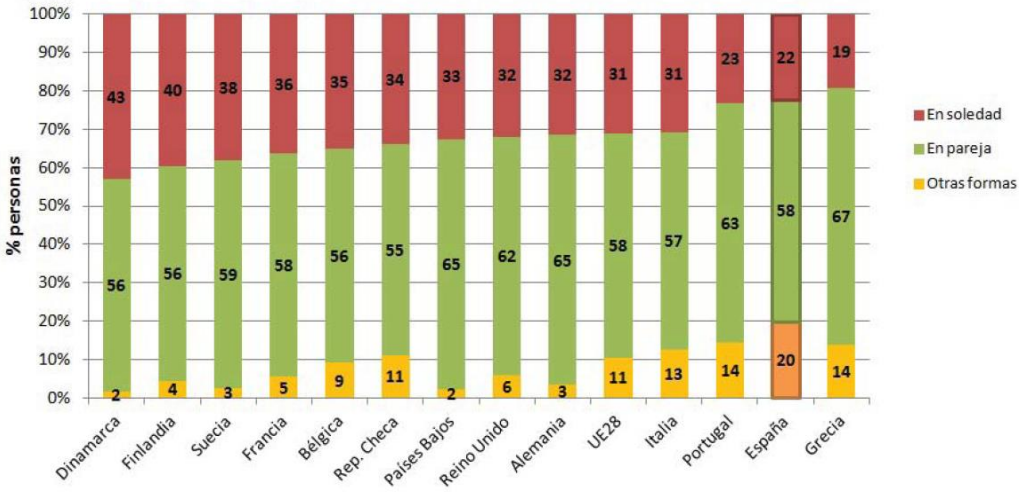



Figura 3. Proyección de la población en España 2014-2064. Fuente: INE

Una vez conocido el continuo envejecimiento de la población, nos podemos fijar en la situación de estas personas. Desde el punto de vista de la forma de convivencia de población de 65 y más años se ha observado que conforme aumenta la edad también aumenta la posibilidad de vivir en soledad, aunque en España nos encontramos en un porcentaje bastante bajo con respecto al resto de países europeos.

En cambio, como también podemos observar en la gráfica de la figura 4, tenemos un porcentaje bastante alto de personas mayores de 65 años que viven en pareja, que es el tipo de convivencia mayoritario entre estas personas.




Fuente: Eurostat. Distribution of population aged 65 and over by type of household (EU-SILC)

Figura 4. Distribución según la forma de convivencia de la población mayor de 65 años en Europa. Fuente: Eurostats

Analizando más profundamente a las personas que viven en soledad, hay una gran diferencia entre hombres y mujeres. Una minoría de hombres viven solos después de los 65 años (un 6%), frente a un 28% de mujeres que lo hacen.

Respecto a la vida en pareja después de los 65 años, el 55% de los hombres desarrollan su vida así, mientras en mujeres el porcentaje baja al 31%.

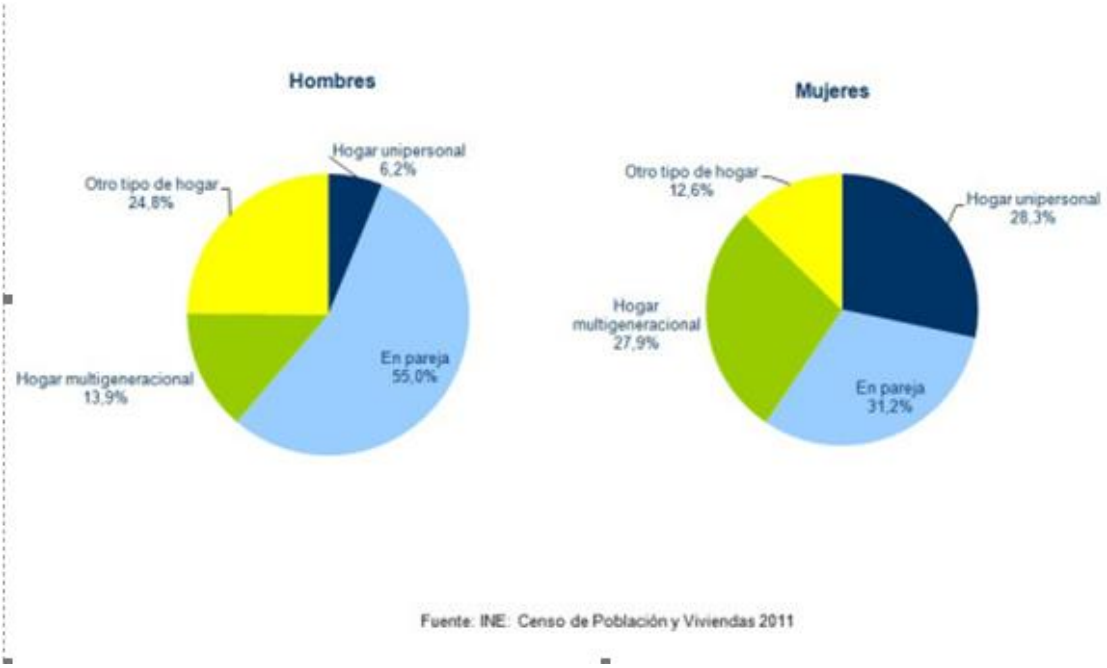



Figura 5. Formas de convivencia segregada por sexos en población mayor de 65 años en el año 2011. Fuente: INE

Si analizamos estos dos datos, segregados por sexo (figura 5), la suma de ambos datos resulta en cerca de un 60% para ambos sexos, lo que significa que la mayoría de casos que nos encontramos en la sociedad es de personas mayores de 65 años viviendo sin personas jóvenes en su hogar.

Teniendo en cuenta estos datos y dada la pérdida de capacidades físicas y psíquicas inherentes al envejecimiento, la sociedad del bienestar se encuentra en la búsqueda de soluciones que palien este problema.

Entre estas soluciones se encuentran las tecnológicas, a las que pertenece el proyecto desarrollado, el cual está compuesto por diferentes sistemas:

- domótico: sistema orientado al control inteligente de la vivienda que permite una gestión que aporta seguridad y confort, además de comunicación entre el usuario y el sistema.

- automático: sistema que modifica y regula el comportamiento de una planta de acuerdo a un protocolo de control preestablecido.

- robótico: sistema organizado capaz de responder con una acción inteligente a los estímulos que es capaz de percibir.

Un gran número de empresas se han sumergido ya en el desarrollo de soluciones para la asistencia a personas mayores puesto que es un mercado que, como ya hemos visto, no hace más que crecer.

Podemos observar que el mercado de robots de teleasistencia es cada vez mayor. Se puede ver que existen robots de compañía, como Asus ZenBo [3], o también podemos ver objetos cotidianos adaptados a las dificultades que se puedan tener, como Robotic Bed, que es una cama articulada que se puede convertir en silla para desplazarse.

En esta línea, por ejemplo, la junta de Andalucía dispone del servicio de teleasistencia, un sistema de atención personalizada que permite a su usuario tener contacto a través de la línea telefónica en caso de una situación de emergencia en cualquier momento y simplemente apretando un botón.

También nos encontramos con numerosos grupos de investigación que están inmersos en el desarrollo de sistemas robóticos, domóticos y automáticos orientados a teleasistencia. Por ejemplo, el proyecto *GIRAFFplus* [4]

*GIRAFFplus* es un sistema complejo que puede monitorizar la actividad que tiene lugar en una casa mediante una red de sensores distribuidos. Los sensores son utilizados para tareas como medir la presión sanguínea del usuario o detectar si éste se ha caído.

Los servicios que provee *GIRAFFplus* pueden ser preseleccionados y adaptados según los requerimientos de la persona mayor o del personal sanitario. En el corazón del sistema se encuentra un robot móvil, el robot *Giraff*, que da nombre al proyecto. El robot utiliza un interfaz similar al de Skype que permite a diferentes usuarios, familiares, cuidadores o personal sanitario, visitar virtualmente a la persona mayor en su propia casa.

## 1.1- MOTIVACIÓN

Como se ha comentado anteriormente el evidente envejecimiento de la sociedad y la mayor esperanza de vida de las personas hace necesario que se desarrollen sistemas que faciliten tener información del estado de la vivienda y del estado del usuario.

El conocimiento del estado del usuario nos permitirá tener un mayor control y, en caso de que sea necesario, poder asistirlo, ayudarlo o dar aviso a servicios de atención médica si es lo que necesita.

Además del estado del usuario respecto a salud, con la disposición de sensores y mediante algoritmia podremos ser capaces de conocer la actividad que está realizando en el momento. Esto nos ayudará a poder confirmar que desarrolla las actividades normales que cualquier persona necesita e incluso, mediante el almacenamiento de ciertos datos poder detectar anomalías en el desarrollo de actividades, como, por ejemplo, el olvido de realizarlas o hacerlas en horas incoherentes.

El conocimiento del estado de los sensores instalados en el domicilio también servirá para que se pueda tener acceso a toda la información disponible en el servidor, que será lanzado por la plataforma que se estudia y se explicará a lo largo de este documento. Estos datos podrán ser vistos y estudiados en tiempo real por cualquier persona que tenga acceso y servirá para conocer el estado de la vivienda en caso de tener algún aviso de emergencia o simplemente por interés o como vigilancia de cualquier servicio de asistencia.

Para poder llevar acabo lo descrito anteriormente se estudiará una plataforma orientada a la automática y domótica.

Se ha decidido estudiar la plataforma *openHAB* porque es un software de código abierto, novedoso y que tiene una gran facilidad de conexión con multitud de tecnologías, lo que nos va a permitir llevar a cabo el proyecto. Nos permitirá realizar desde el control de los sensores, la programación de las actuaciones a realizar a través de la lectura de los anteriores y lanzar una interfaz gráfica a través del navegador para acceder al estado del domicilio de forma visual.

## 1.2- OBJETIVOS

El objetivo de este trabajo fin de grado es el de crear un sistema automático y domótico con el que mediante sensores convenientemente dispuestos en el domicilio de una persona mayor podamos adquirir la información necesaria para inferir qué actividad está realizando el usuario, analizar la situación y actuar en consecuencia. La actuación que podemos realizar será muy diferente dependiendo de la situación, desde ofrecer un mensaje recordatorio en caso de detectar que el usuario ha dejado de hacer alguna

actividad necesaria como la alimentación, la toma de medicamentos, etc., realizar el apagado de algún dispositivo electrónico que el usuario haya dejado encendido y detectemos que no está usando, u ofertar la realización de una actividad diferente, por ejemplo, oír música o llamar a un familiar.

Para cubrir estos objetivos, tendremos que hacer uso de sensores que nos permitan reconocer la actividad del usuario y el estado del entorno en el que se hayan instalado, como por ejemplo detectores de presencia PIR (para saber en qué habitación se encuentra en cada momento), sensores de presión que, situados en sillas o en la cama, nos ayudan a intuir qué actividad se encuentra realizando el usuario, sensores de consumo eléctrico, sensores de apertura/cierre de puerta, etc.

Todos estos sensores estarán conectados a un receptor capaz de registrar los eventos que se produzcan y, conectado a un servidor, poder procesarlos para realizar las acciones de ayuda o apoyo a las que nos referíamos anteriormente.

La toma de decisiones de la acción que debemos realizar será tomada mediante un algoritmo en el que, si detectamos ciertos estados de los sensores durante un tiempo o una combinación de estados de varios sensores, realicemos la acción pertinente. El manejo de la información de los sensores y toma de decisiones se realizará mediante el *standard openHAB*. La plataforma *openHAB* es de código abierto y se está convirtiendo en un standard para soluciones domóticas debido a que es la opción que ofrece una mayor conectividad con tecnologías diferentes como Bluetooth, KNX, TCP/UDP, Dropbox, Google Calendar, MQTT, etc.

Una vez tomada la decisión de qué acción vamos a realizar debemos transmitirla hacia los actuadores que podrán ser desde un altavoz o luces, que indiquen algo al usuario, como un robot móvil que realice una determinada acción, como encender/apagar dispositivos, p.ej. luces, calefacción, televisión, etc.

# Capítulo 2.

# OPENHAB

En este capítulo se presenta la plataforma *openHAB* [5] [6] en la que se ha desarrollado el núcleo de este trabajo de fin de grado. Se describirán los componentes necesarios para su utilización, así como su aplicación a un caso concreto.

## 2.1- QUÉ ES

Cada día tenemos más dispositivos a nuestro alrededor, incluido nuestros domicilios y, aunque todos ellos están dirigidos a mejorar nuestro estilo de vida, normalmente no disponen de un lenguaje común para poder realizar comunicaciones de una forma más sencilla, creando así un sistema realmente automatizado e inteligente en el hogar.

El objetivo de *openHAB* es proporcionar una plataforma de integración para solucionar este problema, proporcionando un protocolo de comunicación común para diferentes dispositivos y sus correspondientes tecnologías.

*openHAB* es un software de código abierto y fácilmente extensible que permite la integración de diferentes sistemas y tecnologías de automatización del hogar en una única solución que permite dominar las reglas de automatización y que ofrece diferentes interfaces de usuario.

*openHAB* puede ejecutarse en cualquier dispositivo que sea capaz de ejecutar una Java Virtual Machine (Linux, Mac, Windows), además, también permite la integración de una gran cantidad de tecnologías diferentes en un mismo proyecto. Dispone de un potente motor de reglas, con una sintaxis sencilla y basada en Java, para satisfacer todas las necesidades de automatización que se tengan.

*openHAB* proporciona diferentes interfaces de usuario basado en Web, así como nativas para iOS y Android.

El proyecto *openHAB* se divide en dos partes. *openHAB runtime* es el paquete que en realidad se ejecutará en el servidor durante el funcionamiento del sistema, y *openHAB designer* que es una herramienta de configuración para el *openHAB runtime*, generación de interfaces de usuario y de reglas.

*openHAB runtime* es un conjunto de paquetes OSGi desplegado sobre una estructura OSGi (Equinox), por tanto, es una solución Java y necesita una JVM para ejecutarse. Basándose en OSGi, proporciona una arquitectura altamente modular, que incluso permite agregar y quitar funcionalidad durante la ejecución sin detener el servicio. La figura 6 ofrece una descripción de los paquetes principales y sus dependencias:

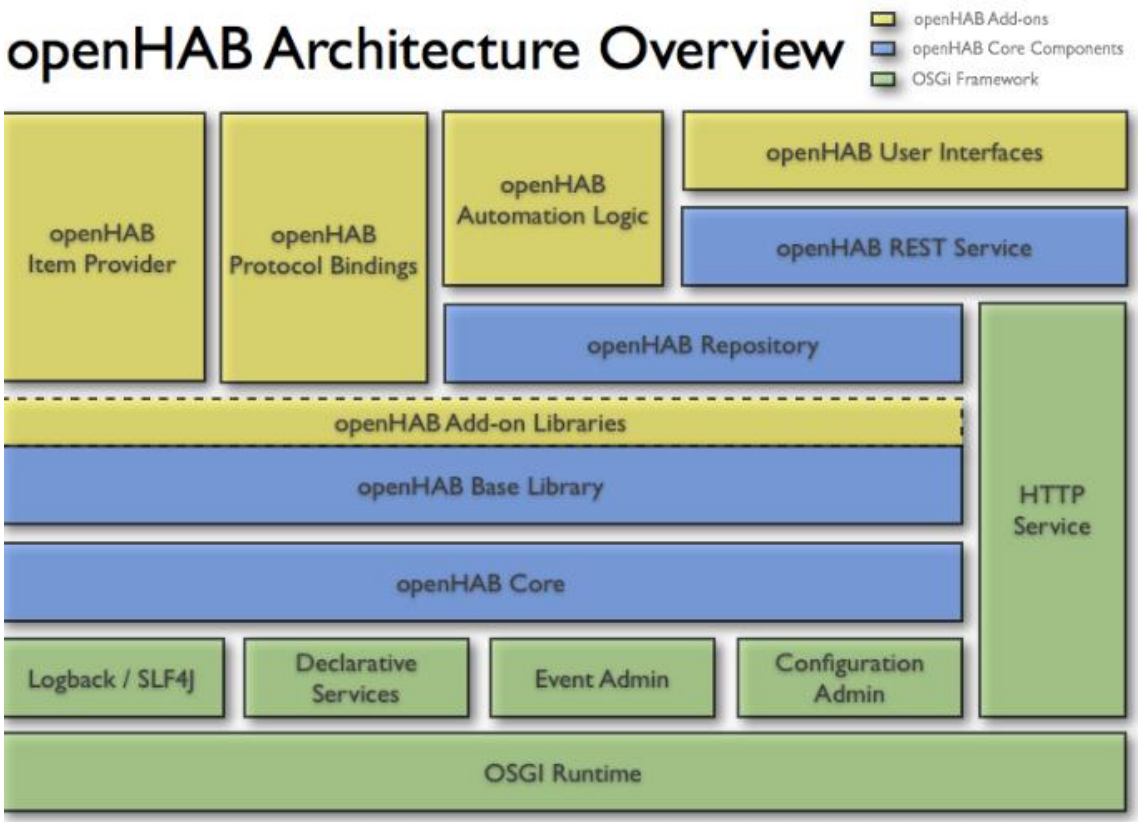



Figura 6. Esquema de la arquitectura de *openHAB*.

*openHAB* dispone de dos canales de comunicación internos. Un bus de eventos asíncrono y un repositorio con estado.

El bus de eventos es el servicio base de *openHAB*. Todos los paquetes que no requieren relación con el estado deben utilizarlo para informar a otros paquetes sobre eventos y para que sean actualizados por otros paquetes por eventos externos.

Hay dos tipos de eventos. Comandos que activan una opción o un cambio de estado de algún elemento/dispositivo y actualizaciones de estado que informan sobre un cambio de estado de algún elemento/dispositivo.

Un *binding* es un paquete opcional que se puede usar para extender la funcionalidad de *openHAB*. Serán los paquetes que nos permitan hacer la conexión con elementos que usen tecnologías diferentes.

Todos los *binding* deben comunicarse a través del bus de eventos, lo que asegura que existe un acoplamiento muy bajo entre los paquetes y facilita la dinámica de *openHAB*.

Es importante tener en cuenta que *openHAB* no está destinado a residir en dispositivos hardware que luego tengan que comunicarse remotamente con otras instancias *openHAB* distribuidas. En su lugar, *openHAB* sirve como centro de integración entre estos dispositivos y como un mediador entre los diferentes protocolos que se comunican entre estos dispositivos. En una instalación, por tanto, solo hay una instancia de *openHAB* en ejecución.

## 2.2- OBJETOS

En este apartado se procederá a la explicación del elemento más básico de *openHAB*, el *Item*.

Un ítem es un objeto que puede ser leído o escrito para interactuar con él. Todos los ítems deben estar definidos en un archivo con la extensión. *items*.

La sintaxis con la que definiremos un ítem será la siguiente:

```
itemTipo itemNombre ["etiqueta"] [nombreIcono] [(grupoA, grupoB, ...)] [configBinding]
```

donde cada una de estas partes de la sintaxis procedemos a explicar a continuación.

### 2.2.1- TIPOS DE ITEMS

El tipo de ítem define qué tipo de valores se pueden almacenar en ese elemento y qué comandos se pueden enviar a él. Cada tipo de elemento se ha optimizado para ciertos componentes en el hogar inteligente.

Los tipos que pueden definir a un ítem son:

| Tipo del ítem | Descripcion | Tipos de datos aceptados | Tipos de comandos aceptados |
|---|---|---|---|
| Call (llamada) | Este tipo puede ser usado para elementos relacionados con las funcionalidades de telefonía. | Call | - |
| Color | Puede ser usado para valores de color | On/Off, porcentaje, HSB | On/Off, incremento/decremento, porcentaje, HSB |
| Contact (contacto) | Puede ser usado en elementos que tengan como estado “abierto” o “cerrado” | Open/Closed | |
| DateTime (fecha) | Almacena un dato de tiempo | DateTime | DateTime |
| Dimmer | Acepta porcentajes para | On/Off, porcentajes | On/Off, incremento/decremento, porcentajes |

| | | | |
|---|---|---|---|
| | establecer el valor. | | |
| Group (grupo) | Colección de ítems | Acepta el tipo de datos de los ítems a los que hace referencia | Acepta el tipo de comandos de los ítems a los que hace referencia |
| Location (localización) | Puede ser utilizado para almacenar información relacionada con GPS, direcciones, etc | Coordenadas | Coordenadas |
| Number (numero) | Puede ser usado para cualquier tipo de sensor numérico como temperatura, brillo, viento, etc o como un contador. | Numero | Numero |
| Rollershutter (persiana) | Permite el control de las persianas moviendo hacia arriba, abajo y parando o estableciendo un porcentaje | Up/Down, porcentaje | Up/Down, StopMove, porcentaje |
| String (cadena de caracteres) | Puede ser usado por cualquier tipo de cadena de texto o representación textual de una fecha | String, fecha | String |
| Switch (interruptor) | Representa a cualquier elemento que puede estar en estado ON o en estado OFF | On/Off | On/Off |

## 2.1.2- GRUPOS

Una de los aspectos más útiles a explicar es el concepto de grupo de los ítems. En caso de que resulte necesario o que nos sea de interés, podemos agrupar diferentes

ítems en un mismo grupo, esto nos servirá para poder hacer referencia a todos estos ítems en cualquier momento simplemente haciendo referencia a su grupo.

Así, nos podremos referir al grupo y asignar funciones a éste estableciendo un valor si todos los elementos de este grupo son del mismo tipo. Las funciones que podremos usar son las operaciones lógicas AND, NAND, NOR y OR, AVG (media), MAX (máximo), MIN (mínimo) y SUM (sumatorio).

### 2.1.3- NOMBRE DEL ITEM

El nombre de un ítem es el nombre único de un elemento. El nombre debe ser único ya que será el que utilicemos en otras secciones (reglas o IU) para referirnos al elemento en cuestión.

### 2.1.4- ETIQUETAS

Las etiquetas en *openHAB* tienen dos propósitos, primero describir de una forma más natural que el nombre del ítem el elemento para, por ejemplo, que sea reconocible en la interfaz gráfica. En segundo lugar, se puede utilizar para dar formato a la salida del elemento, es decir, transformar el dato de la forma que nos sea más conveniente, esto se podrá hacer de la misma forma que en Java.

### 2.1.5- NOMBRE DEL ICONO

El nombre del icono se utiliza para hacer referencia a un archivo de imagen png de una carpeta del directorio *openHAB*. Estos iconos se utilizan en los *front-ends* y *openHAB* proporciona un conjunto básico a los que se pueden añadir los que se deseen.

### 2.1.6- CONFIGURACIONES DE BINDINGS

La instalación de un *binding* es tan sencilla como copiar en la carpeta correspondiente del directorio de *openHAB* los *binding* que podemos descargar de la página de *openHAB*.

La configuración de *binding* es la parte más importante de un elemento, ya que define donde el elemento obtiene valores y donde se deben enviar los valores relacionados con él. Como cada *binding* tiene una configuración diferente se debe consultar su documentación donde se detallan las configuraciones pertinentes a realizar.

## 2.3- REGLAS

Las reglas en *openHAB* son usadas para automatizar procesos mediante la ejecución de un script y un motor de reglas altamente integrado y potente.

Las reglas deberán estar en un archivo con extensión .rules en el directorio de *openHAB*. Un archivo de reglas puede contener varias reglas y todas ellas comparten un contexto de ejecución común, es decir, pueden acceder e intercambiar variables entre sí. Por este motivo puede tener sentido tener diferentes archivos de reglas para diferentes casos de uso.

El IDE que nos brinda *openHAB* tiene un soporte completo para las reglas que incluye controles de sintaxis y coloreado del texto, validación con marcadores de error, asistencia de contenido, plantillas, etc. para facilitar la creación de reglas (figura 7).

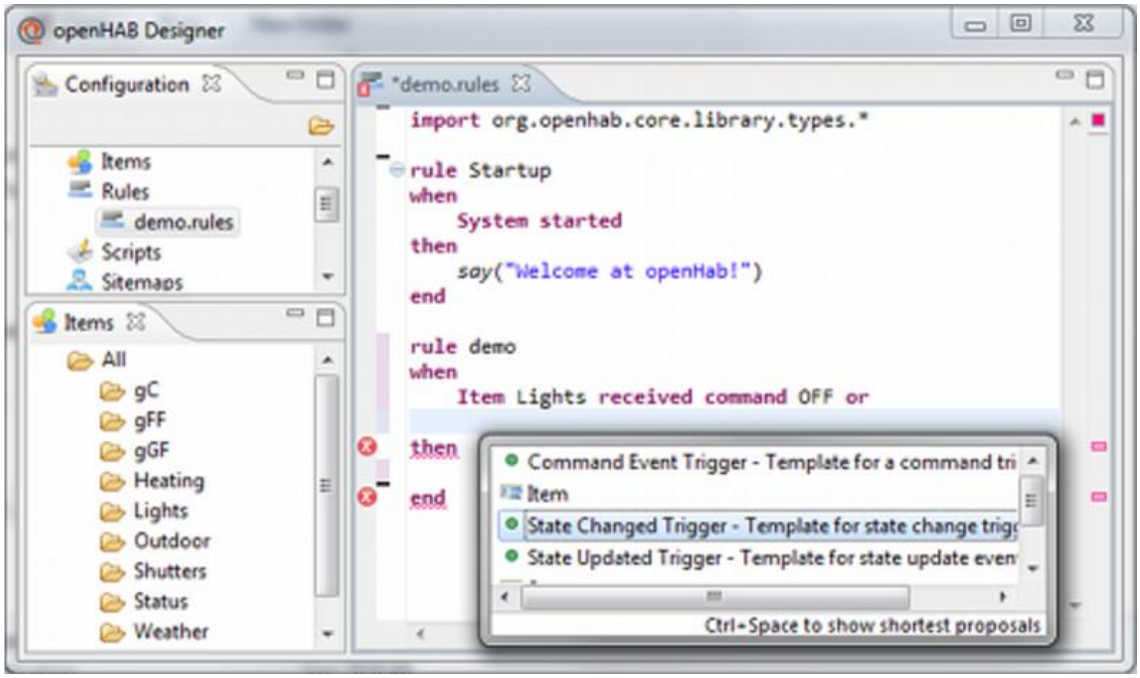



Figura 7. Captura de pantalla de *openHAB designer*.

La sintaxis que debe seguir un archivo de reglas es: primero las importaciones, seguido de las variables y por último la definición de la lógica de la regla. Las importaciones se realizan de igual forma que en Java y hacen que dispongamos más fácilmente de los tipos importados. En la declaración de variables se usará para declarar todas las variables que deben ser accesibles para todas las reglas de este archivo; pueden ser declaradas con o sin valores iniciales. Las reglas pueden tener cualquier número de condiciones, pero al menos debe tener una que, cuando se cumpla, hará que se ejecute el script que deseemos.

Antes de que una regla empiece a funcionar debe ser activada. La activación de las reglas se hace por disparadores y hay tres tipos:

- Disparadores basados en *Item*: reaccionan sobre eventos en el bus de eventos de *openHAB*, es decir, comandos y actualizaciones de estado de los ítems.
- Disparadores temporales: reaccionan en momentos especiales que sean interesantes, horarios habituales (mediodía, noche), en una hora concreta o cada cierto tiempo.
- Disparadores basados en el sistema: reaccionan cuando el sistema se encuentra en un estado determinado.

## 2.3.1- BASADAS EN ITEMS

Este tipo de reglas puede procesar comandos de un elemento especifico, actualizaciones de estado o cambios de estado (una actualización puede dejar el estado sin cambio). Se puede decidir si se desea capturar solo un comando/estado especifico o cualquiera.

La primera opción con la que nos encontramos es que un ítem reciba un comando (figura 8).

```
rule "item recibe comando"
    when
        Item itemName received command ON
    then
    ...
end
```

Figura 8. Ejemplo de regla disparada al recibir un comando.

Otra opción es que el ítem reciba una actualización (figura 9).

```
rule "item recibe actualizacion"
    when
        Item itemName received update ON
    then
    ...
end
```

Figura 9. Ejemplo de regla disparada al recibir una actualización del ítem.

Otro tipo de disparador es que haya un cambio en el estado del ítem (figura 10).

```
rule "item cambia el estado"
    when
        Item itemName changed
    then
    ...
end
```

Figura 10. Ejemplo de regla disparada al recibir un cambio en el estado del ítem.

Y el ultimo tipo de disparador está relacionado con el anterior y sería un cambio, pero especifico de un estado “A” a otro estado “B” (figura 11).

```
rule "item cambia del estado On a OFF"
    when
        Item itemName changed from ON to OFF
    then
    ...
end
```

Figura 11. Ejemplo de regla disparada al recibir el ítem un cambio entre dos estados específicos.

Todos estos tipos de disparadores pueden ser conjuntados en caso de que nos interesen varios tipos con el operador lógico ‘*or*’.

## 2.3.2- BASADAS EN EL TIEMPO

Para los disparadores basado en el tiempo se pueden usar expresiones ya predefinidas en el sistema o utilizar una expresión tipo ‘*cron*’ [7] [8] en su lugar.

Como expresiones predefinidas podemos encontrar medianoche o mediodía (figura 12).

```
rule "basadas en el tiempo"
    when
        Time is midnight
    then
    ...
end
```

Figura 12. Ejemplo de regla disparada por expresión predefinida.

Respecto a las expresiones de tiempo son expresiones que tienen siete campos: segundos, minutos, horas, día del mes, mes, día de la semana y año. Con estas expresiones podemos establecer cualquier tramo horario, fecha concreta, etc.

```
rule "basadas en el tiempo"
    when
        Time cron " 0 0 8 ? * MON-FRI *"
    then
    ...
end
```

Figura 13. Ejemplo de regla disparada por expresión temporal.

La expresión temporal de la regla de la figura 13 será para una regla que se disparará todos los días entre semana a las 8 de la mañana.

### 2.3.3- BASADAS EN EL ESTADO DEL SISTEMA

Las reglas más importantes, en cuanto a planificación se refiere, siempre se realizan en el inicio (figura 14) o apagado (figura 15) del sistema y estos disparadores se encuentran en este apartado. Estas reglas solo se ejecutarán una vez, cuando el sistema detecta que está en uno de los dos estados descritos.

```
rule "basadas en el sistema"
    when
        System started
    then
    ...
end
```

Figura 14. Ejemplo de regla disparada cuando el sistema esté iniciándose.

```
rule "basadas en el sistema"
    when
        System shuts down
    then
    ...
end
```

Figura 15. Ejemplo de regla disparada cuando el sistema esté apagándose.

## 2.4- SITEMAPS

Un *sitemaps* es un modelo visual o textualmente organizado del contenido de un sitio web que permite a los usuarios navegar a través del sitio para encontrar la información que buscan.

En *openHAB*, el termino *sitemaps* se utiliza para referirse al documento que enumera el contenido de su sistema de automatización que se mostrará en la pantalla del usuario.

El *sitemaps* será la interfaz de usuario que tendremos en el navegador para poder ver el estado de los elementos que seleccionemos.

Los elementos que podremos utilizar son los siguientes:

| Elemento | Descripción |
|---|---|
| Colorpicker (selector de color) | Permite al usuario elegir el color |
| Chart | Agrega un objeto grafico para mostrar los datos registrados |
| Frame (Área) | Área con otros elementos de sitemap o marcos anidados adicionales |
| Group (grupo) | Agrupa los elementos definidos en un grupo |
| Image (imagen) | Muestra una imagen |
| Selection (selección) | Da acceso a una nueva página donde el usuario puede elegir entre los valores definidos. |
| Setpoint | Muestra un valor y permite al usuario modificarlo. Se pueden especificar valores máximos y mínimos. |
| Slider (deslizador) | Control deslizante |
| Switch (interruptor) | interruptor |
| Text (texto) | Texto |
| Video (video) | Muestra un video |
| Webview (visor web) | Muestra una página web |

*openHAB* da la opción de establecer diferentes interfaces gráficas (figuras 16, 17, 18, 19 y 20) para el usuario además de tener interfaces gráficas para plataformas tan importantes como Android y iOS.


Figura 16. Imagen de la UI Android. Fuente: http://www.openhab.org.

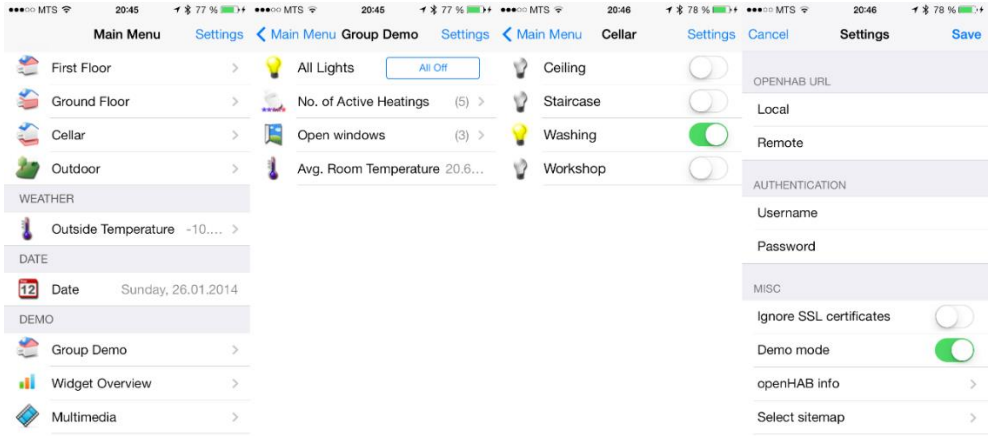



Figura 17. Imagen de la UI iOs. Fuente: http://www.openhab.org.


Figura 18. Imagen de la UI Classic. Fuente: http://www.openhab.org.

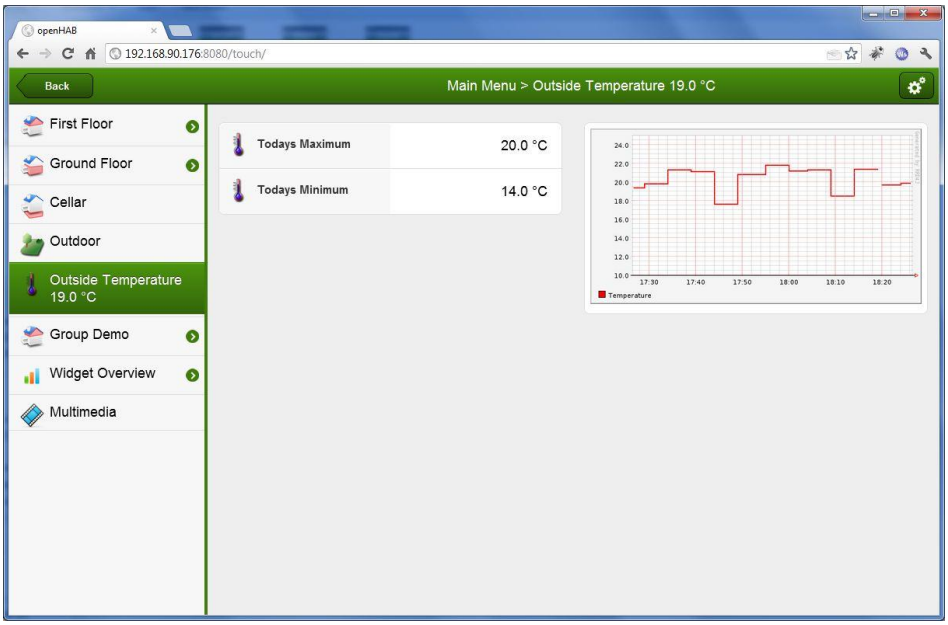



Figura 19. Imagen de la UI GreenT. Fuete: http://www.openhab.org.

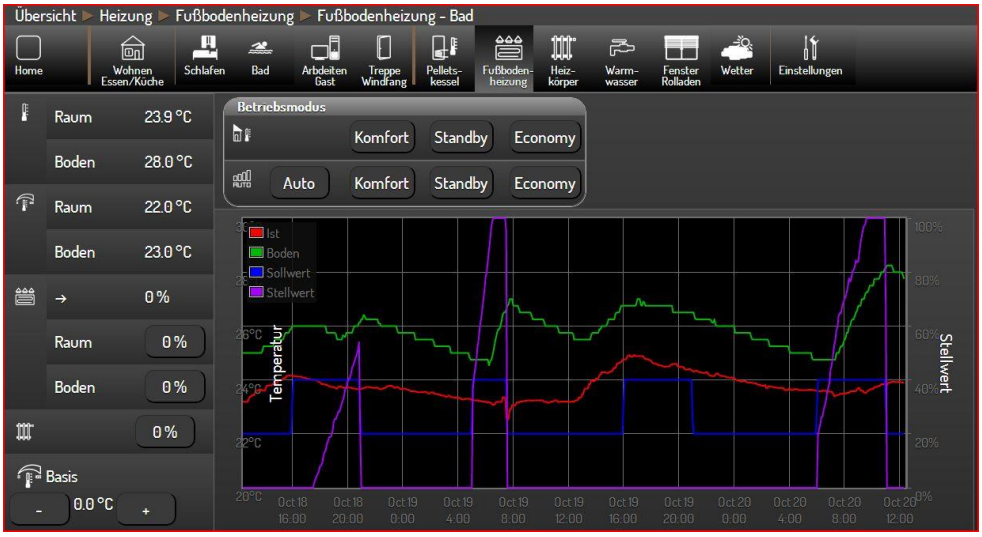



Figura 20. Imagen de la IU CometVisu. Fuente: http://www.openhab.org.

## 2.5- ARCHIVO DE CONFIGURACIÓN

Este apartado está relacionado con el archivo con extensión '*.cfg*'. Este archivo es el que nos permite establecer la configuración de la mayoría de *addons* que podemos integrar en el proyecto. Es una plantilla comentada en la que están almacenadas las variables de los datos que son necesarias para el uso de los *addons* que nos proporcionan. En caso de necesitar usar un *addons*, se incluirá en la carpeta correspondiente y tendremos que modificar este archivo para introducir los datos que sean necesarios para el correcto funcionamiento.

## 2.6- ACTIONS

En este apartado se procede a explicar las acciones más relevantes relacionadas con las distintas opciones que se nos ofrece.

### 2.6.1- ACCIONES RELACIONADAS CON EL BUS DE EVENTOS

- *sendCommand (String itemName, String commandString)*: envía el comando dado para el ítem especificado al bus de eventos.
- *postUpdate(String itemName, String stateString)*: envía el estado dado al ítem especificado por el bus de eventos.
- *Group.send(command command_to_send)*: envía el comando dado a todos los elementos del grupo
- *Map<ítem, state> storesStates(Item…ítems)*: almacena el estado actual de una lista de elementos en u mapa que se puede asignar a una variable.

- *restoreStates(Map<ítem, state> statesMap)*: restaura estados de elementos de un mapa. Si el estado guardado se puede interpretar como un comando, se envía un comando para el elemento.

### 2.6.2- TIMERS

- *createTimer(AbstractInstant instant, Procedure procedure)*: programa un bloque de código para la ejecución segura.

  Este método devuelve un objeto tipo *Timer*, que soporta los siguientes métodos.

  - *Cancel()*: cancela el *timer*.
  - *isRunning()*: devuelve true si el código programado está actualmente ejecutando.
  - *hasTerminated()*: devuelve true si el código programado ha terminado de ejecutarse.
  - *reschedule(AbstractInstant newTime)*: restablece un nuevo tiempo de inicio, si el temporizador ha terminado, este método no hace nada.

### 2.6.3- ACCIONES DE CORREO ELECTRÓNICO

Este complemento proporciona servicios SMTP. La configuración necesaria se implementará en el archivo de configuraciones con extensión '*.cfg*'.

- *sendMail(String to, String subject, String message)*: envía un correo electrónico vía SMTP.
- *sendMail(String to, String subject, String message, String attachmentUrl)*: envía un correo electrónico con datos adjuntos vía SMTP.
- *sendMail(String to, String subject, String message, List<String> attachmentUrlList)*: envía un correo electrónico con uno o más archivos adjuntos vía SMTP.

### 2.6.4- ACCIONES MQTT

- *publish(String brokerName, String topic, String message)*: publica el mensaje en el tema utilizando el broker MQTT especificado.

### 2.6.5- OTRAS

- *executeCommandLine(String commandLine)*: ejecuta un comando en la línea de comandos.

# Capítulo 3.

# HARDWARE

En este apartado se va a proceder a la descripción de los diferentes elementos hardware usados en el proyecto.

Primeramente, deberemos diferencias los dos grandes grupos en los que se pueden dividir estos elementos, sensores y actuadores.

## 3.1- TUNSTALL LIFELINE

El *Lifeline Vi* [9] (figura 21) proporciona un centro de teleasistencia en el hogar, compatible con una amplia gama de sensores que se pueden elegir de acuerdo a las necesidades del usuario.


Figura 21. Vista superior de TUNSTALL *Lifeline*.

Este elemento será el principal receptor de nuestro sistema, ya que será el encargado de recibir todos los cambios que los sensores detecten. Además, tendrá que realizar la comunicación de estos cambios al servidor en el que se esté ejecutando *openHAB*.

## 3.2- SENSORES

Un sensor es un objeto capaz de detectar magnitudes físicas o químicas, llamadas variables de instrumentación, y transformarlas en variables eléctricas.

Los sensores que se han usado en este proyecto son todos de la marca *Tunstall* y se describen a continuación.

## 3.2.1- SENSOR UNIVERSAL/PUERTA

El Sensor Universal [10] (figura 22) permite a los dispositivos cableados transmitir una señal a una unidad doméstica *Lifeline*, eliminando la necesidad de conectar el sensor o el detector al sistema. Permite a los dispositivos cableados y otros equipos transmitir las llamadas de alarma inalámbrica y los mensajes de radio apropiados a la unidad doméstica mediante la funcionalidad Plug and Play. También puede usarse como un sensor de uso de la puerta utilizando los contactos de puerta suministrados.


Figura 22. Sensor Universal o de puerta.

El sensor universal funciona como una interfaz entre dispositivos eléctricos cableados y sistemas de Telecare de *Tunstall* habilitados. Estos pueden incluir un sistema de alarma de intrusos existente, detector de gas natural, detector de monóxido de carbono o estera de presión. Los interruptores en la parte posterior del sensor permiten que el instalador fije fácilmente el sensor para enviar un mensaje a la unidad central que identifica qué tipo de dispositivo se liga. Cuando el dispositivo conectado es activado, el sensor universal envía el mensaje apropiado, proporcionando al operador en el centro de respuesta suficiente información para responder en consecuencia. El sensor universal también puede recibir entradas de contactos normalmente abiertos o normalmente cerrados, permitiendo su conexión a una amplia gama de dispositivos.

## 3.2.2- SENSOR PRESIÓN

El sensor de ocupación de cama / silla [11] (figura 23) está diseñado para detectar presión. Se puede programar para supervisar la actividad o inactividad, y se puede utilizar para detectar situaciones de interés para los usuarios, por ejemplo:
• no se ha ido a la cama en un tiempo predeterminado
• no ha salido de la cama en un tiempo predeterminado
• se levanta de la cama durante la noche y no regresa dentro de un período de tiempo determinado.

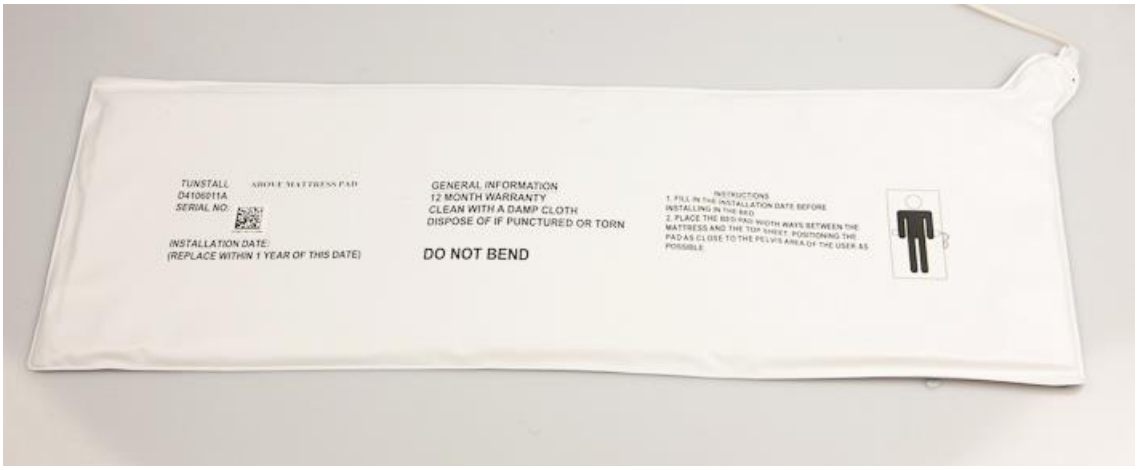



Figura 23. Sensor de presión.

La solución de ocupación de cama / silla puede adaptarse a las actividades diarias específicas de un usuario individual.

El sensor de ocupación de cama / silla consiste en una almohadilla de presión, que se coloca debajo del colchón del cliente o en su silla.

## 3.2.3- SENSOR CORRIENTE

El sensor de uso eléctrico (figura 24) envía una señal de radio a la unidad familiar *Tunstall Lifeline* cuando se conecta/desconecta un aparato eléctrico. Está diseñado para ser utilizado como parte de los sistemas *ADLife* de *Tunstall* para indicar cuando los usuarios de teleasistencia han utilizado un aparato en particular, como una tetera, una tostadora o un microondas. También se puede utilizar como una entrada adicional para el monitoreo de inactividad.

Figura 24. Sensor de corriente.

## 3.3 ACTUADORES

De forma general, un actuador es un dispositivo capaz de transformar energía hidráulica, neumática o eléctrica en la activación de un proceso con la finalidad de generar un efecto sobre un proceso automatizado. Éste recibe la orden de una regulador o controlador y en función a ella genera la orden para activar un elemento final de control.

En este trabajo fin de grado se han considerado tres tipos de actuadores que permiten modificar el estado del entorno en base a la información proporcionada por los sensores y a las reglas implementadas.

### 3.3.1- ORVIBO

*Orvibo* [12] (figura 25) es un dispositivo equipado con un emisor IR de gran tamaño situado en la parte superior del dispositivo, que cubre hasta 360º, facilitando así la comunicación con cualquier aparato IR situado en su campo de visión.

Dispone de un único botón central con el que se podrá realizar la configuración siempre ayudándonos de la App que tendremos que instalar en un dispositivo Android o iOS.

Figura 25. Imagen de *Orvibo*

## 3.3.2- IP POWER WIFI

IP *Power* WIFI [13] (figuras 26 y 27) es un elemento con capacidad para conectar cuatro dispositivos eléctricos y que dispone de conexión WIFI a través de la cual se puede comunicar con él para activar o desactivar cada una de sus entradas.


Figura 26. Vista frontal IP *Power* WIFI.


Figura 27. Vista trasera IP *Power* WIFI.

## 3.3.3- ROBOT MÓVIL SANCHO

El dispositivo móvil que se ha usado para realizar actividades de prueba es el robot de servicios SANCHO [14] (figura 28). Este dispositivo tiene las siguientes características:

- Base robótica ActivMedia Pioneer P3DX
- Procesador Intel i5 - 650@3.20GHz
- Disco duro SSD de 250GB
- 4 GB de memoria RAM
- Windows 10 Pro y Ubuntu 14.04 (para ROS, no usado)
- OpenMora (Robotic Software) + Tunstall Library
- Google API TTS (Text-To-Speech)
- Anillo de sónares (en la base -- no usados)
- Bumpers frontales (en la base -- no usados)
- 2 Escáneres láser Hokuyo UTM-30LX (para localización)
- 1 Cámara estéreo Bumblebee2 BB2-S802C (no usada)
- 1 Cámara RGB-D (Kinect) (no usada)
- 1 Cámara GigE Monochrome con zoom (no usada)
- 1 Webcam Live! Cam 720HD (detección y reconocimiento de caras)
- Servomotor para control de giro del cuello
- Sistema de iluminación del habla

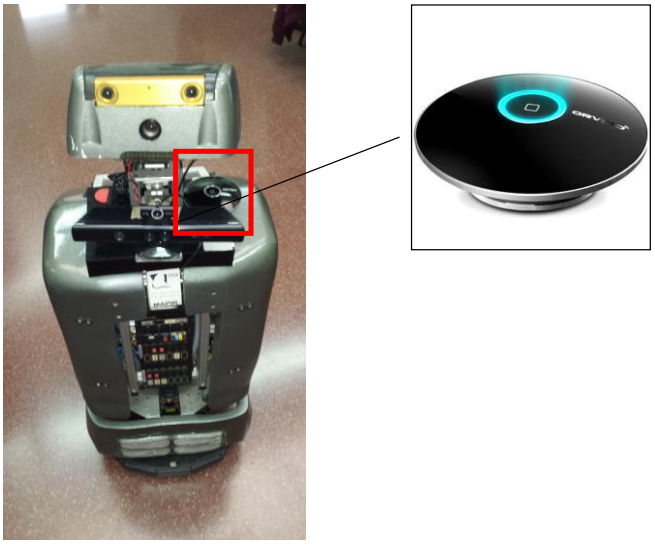


Figura 28. SANCHO Service Robot

El robot SANCHO está controlado por la arquitectura de control robótica OpenMora [15] que le permite desplazarse por su entorno, esquivar los obstáculos presentes ya sean estáticos o dinámicos e interactuar con personas de sus alrededores mediante

voz. En este trabajo se ha utilizado al robot SANCHO como portador del dispositivo *Orvibo*, con el que poder comandar diferentes dispositivos vía IR.

# Capítulo 4.

# IMPLEMENTACIÓN

En este apartado se va a proceder a la explicación de la implementación realizada en el proyecto.

## 4.1- COMUNICACIONES

Una vez realizado el montaje respecto a lo que a hardware se refiere, con los sensores conectados con *Tunstall Lifeline*, se procederá a la comunicación de los datos recibidos al servidor donde tengamos instalado *openHAB* para poder hacer tratamiento de estos datos.

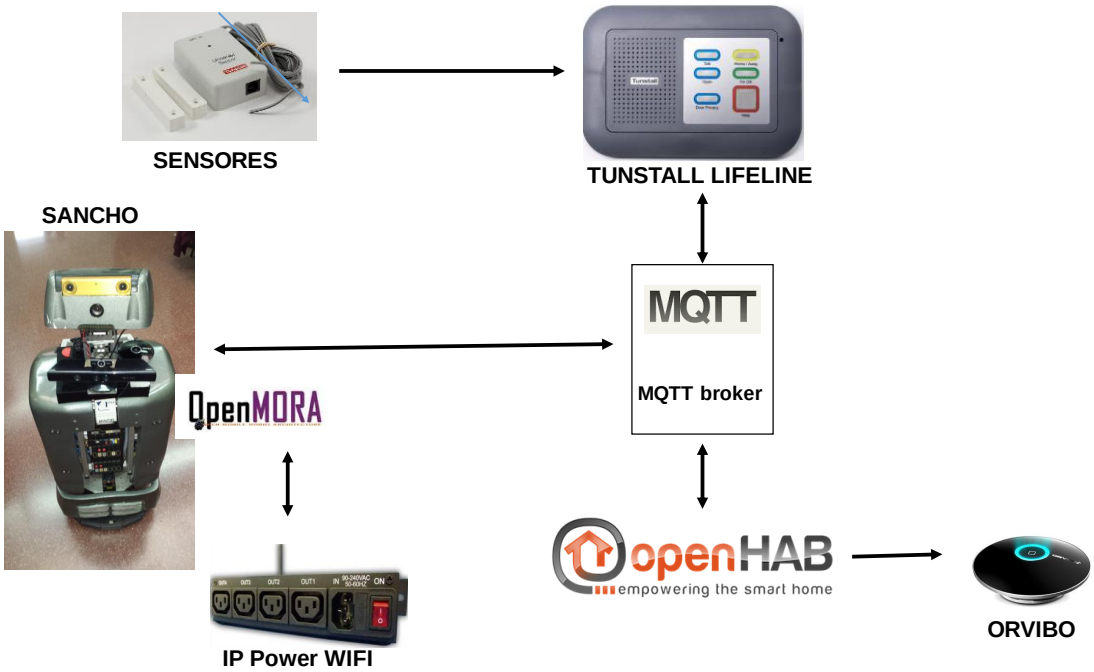



Figura 29. Esquema de conexiones del sistema.

Esta comunicación se realizará a través de un puerto USB con el servidor. Para realizar la comunicación usamos *OpenMORA* .

*OpenMORA* es una arquitectura de control robótica híbrida mantenida por el laboratorio MAPIR (Universidad de Málaga), el Laboratorio de Ingeniería Mecánica (Universidad de Almería) y colaboradores individuales. OpenMora proporciona módulos estándar para plataformas y sensores comunes de robótica, registro de conjuntos de datos, localización *MonteCarlo*, navegación reactiva y planificada, etc.,

en base a la *Mobile Robot Programming Toolkit* [16] *(MRPT)*. MRPT proporciona a los desarrolladores aplicaciones y bibliotecas portátiles y bien probadas que cubren estructuras de datos y algoritmos empleados en áreas comunes de investigación robótica. Es de código abierto, publicado bajo la licencia BSD. MRPT comprende un conjunto de bibliotecas C ++ y una serie de aplicaciones listas para usar, así como aplicaciones GUI utilizando la biblioteca multiplataforma *wxWidgets*. Además, el sistema multiplataforma *CMake* se utiliza para generar "*Makefiles*" o "proyectos" para cada plataforma según los requisitos del usuario y las bibliotecas presentes en cada sistema.

Para la comunicación hasta la plataforma *openHAB* y de ésta hacia los actuadores usamos *MQTT (Message Queue Telemetry Transport)*, un protocolo usado para la comunicación machine-to-machine (M2M) en el "*Internet of Things*"**.** Este protocolo está orientado a la comunicación de sensores, debido a que consume muy poco ancho de banda y puede ser utilizado en la mayoría de los dispositivos empotrados con pocos recursos (CPU, RAM, …). El funcionamiento del sistema de comunicación MQTT requiere del uso de un servidor o *broker* al que se conectan los distintos dispositivos mediante un sistema de suscripción a unos determinados *topics*. Cuando el servidor recibe datos asociados a un determinado topic, los reenvía a aquellos dispositivos que están suscritos a dicho topic. Existen multitud de servidores MQTT en el mercado que son de uso gratuito.

De entre ellos *Eclipse Mosquitto* [17] proporciona una implementación de servidor ligero del protocolo MQTT que es adecuado para todas las situaciones desde máquinas de máxima potencia hasta máquinas empotradas y de baja potencia. Los sensores y actuadores, que son a menudo las fuentes y destinos de los mensajes MQTT, pueden ser muy pequeños y carentes de potencia. Esto también se aplica a las máquinas embebidas a las que están conectados, que es donde *Mosquitto* podría ser ejecutado.

Además de aceptar conexiones desde aplicaciones cliente MQTT, *Mosquitto* tiene un puente que le permite conectarse a otros servidores MQTT, incluyendo otras instancias de *Mosquitto*. Esto permite construir redes de servidores MQTT, pasando mensajes MQTT desde cualquier lugar de la red a cualquier otro, dependiendo de la configuración de los puentes.

Por último, la comunicación con *Orvibo* es la única diferente ya que usamos un módulo en *python* [18] que nos permite la comunicación a través de la red en la que esté conectado el dispositivo y que tiene que ser la misma que el servidor.

El módulo nos permite a través de la línea de comandos realizar las siguientes operaciones:

- *orvibo.py*: nos devolverá los dispositivos *Orvibo* detectados en la red, su ip y su mac.

- *orvibo.py* ***-i ip***: nos devolverá la información del dispositivo y si está disponible.
- *orvibo -i ip* ***-t test.ir***: creara un archivo con la información del código de infrarrojos que ha recibido
- *orvibo.py -i ip* ***-e test.ir***: emitirá el código de infrarrojos guardado en el archivo test.ir

## 4.2- REGLAS

En este apartado se van a mostrar algunas de las reglas creadas para el uso tanto de la monitorización de la actividad de personas mayores como de la domotización de un domicilio.

- Un aspecto importante de la monitorización sería saber si el usuario está realizando una actividad en la que vaya a transcurrir un tiempo largo. El ejemplo que se expone en las figuras 30 y 31 es la visualización de la televisión y, en caso de que esta actividad se realice por más de una hora, se le ofrecerá una nueva actividad.

```
String mqttSensorPresion     {mqtt="<[mosquitto:sancho/amI/
            tunstall/mapir/lab236/chair/01:state:default]"}
String mqttUsoElectricoTV    {mqtt="<[mosquitto:sancho/amI/
            tunstall/mapir/lab236/elec_usage/01:state:default]"}
String mqttOutMessage        {mqtt=">[mosquitto:sancho/Debug
            :command:*:default]"}
```

Figura 30. Declaración de *Items*.

```
rule "viendo TV"
    when
        Item mqttSensorPresion changed from OFF to ON
    then
        if (mqttUsoElectricoTV.state==1){
            timer = createTimer(now.plusSeconds(3600)) [|
                mqttOutMessage.sendCommand("NEW_TASK " +
                    userID.toString + " " + taskID.toString + " MOVE CARLOS"
                )
                taskID = taskID+1
                mqttOutMessage.sendCommand("NEW_TASK " +
                    userID.toString + " " + taskID.toString + " SAY_MQTT
                    mensaje para ofreces nueva actividad"
                )
                taskID = taskID+1
            ]
        }
end
```

Figura 31. Código de ejemplo.

- Una regla interesante respecto al ahorro de energía seria que, si el usuario va a estar situado en una habitación en concreto, podamos apagar las luces del resto del domicilio para ahorra energía (figuras 32 y 33).

```
Group:Switch:OR(ON, OFF)        Luces               "Luces [(%d)]"
Switch Luz_TV_Salon             "TV"                    (SalonComedor, Luces)
Switch Luz_Comedor              "Comedor"               (SalonComedor, Luces)
Switch Luz_Cocina               "Cocina"                (Cocina, Luces)
Switch Luz_Bano                 "Baño"                  (Bano, Luces)
Switch Luz_Habitacion           "Habitacion"            (Habitacion, Luces)
```

Figura 32. Declaración de *Items*.

```
rule "sentado en salon"
    when
        Item mqttSensorPresion changed from OFF to ON
    then
        Luces?.members.forEach(light|
            postUpdate(light, OFF)
        )
end
```

Figura 33. Código de ejemplo.

- A cierta hora del día puede que sea habitual la realización de una acción, como ejemplo bajar las persianas a media noche, que se puede combinar con acciones que permitan tener cierta seguridad, como apagar el brasero (figuras 34 y 35).

```
Rollershutter   PersianaSalon1
Rollershutter   PersianaSalon2
Rollershutter   PersianaCocina
Rollershutter   PersianaHabitacionPequena
Rollershutter   PersianaHabitacion
Rollershutter   PersianaHabitacionGrande
```

Figura 34. Declaración de Items.

```
rule "son las 00.00h"
    when
        Time is midnight
    then
        //apagamos el brasero (conectado al puerto 0
        mqttOutMessage.sendCommand ("NEW_TASK 123 0 SWITCH_OFF 0")
end
```

Figura 35. Código de ejemplo.

- Respecto a reglas temporales se puede programar una alarma durante todos los días de la semana (figura 36) o realizar un recordatorio en una fecha específica (figura 37).

```
rule "alarma"
    when
        Time cron "0 30 9 1/1 * ? *" //todos los dias a las 9.30
    then
        mqttOutMessage.sendCommand("NEW_TASK " +
            userID.toString + " " + taskID.toString + " MOVE CARLOS")
        taskID = taskID+1
        mqttOutMessage.sendCommand("NEW_TASK " +
            userID.toString + " " + taskID.toString + " SAY_MQTT
            mensaje para despertar")
        taskID = taskID+1
end
```

Figura 36. Código de ejemplo de alarma

```
rule "recordatorio"
    when
        Time cron " 0 0 10 20 4 ? *" //20 abril a las 10 // 0 30 11 ? * TUE,THU,SAT *
    then
            mqttOutMessage.sendCommand("NEW_TASK " +
                userID.toString + " " + taskID.toString + " MOVE CARLOS")
            taskID = taskID+1
            mqttOutMessage.sendCommand("NEW_TASK " +
                userID.toString + " " + taskID.toString + " SAY_MQTT
                mensaje de recordatorio")
            taskID = taskID+1
end
```

Figura 37. Código de ejemplo de recordatorio (20 de abril a las 10.00 AM).

- Se podría actuar también como recordatorio de actividades necesarias, como comer o tomar medicamentos. En el ejemplo se haría con la combinación de dos reglas: una realizará el control de la puerta del frigorífico (figura 39), mientras la otra regla, basándose en la primera que hemos comentado, realizará el aviso cada media hora (figura 40).

```
Switch RepetirAvisoComida
```

Figura 38. Creación de *switch* virtual que nos permite disparar una regla.

```
rule "has comido? puerta frigorifico"
    when
        Item PuertaFrigorifico changed
    then
        puertaFrigorificoAbierta = true
end
```

Figura 39. Código ejemplo de control de uso de frigorífico.

```
rule "has comido? desde 13.30"
    when
        Time cron "0 30 13 1/1 * ? *" or
        Item RepetirAvisoComida received command
    then
        mqttOutMessage.sendCommand("NEW_TASK " +
            userID.toString + " " + taskID.toString + " MOVE CARLOS")
        taskID = taskID+1
        mqttOutMessage.sendCommand("NEW_TASK " +
            userID.toString + " " + taskID.toString + " SAY_MQTT
            mensaje para recordar comer")
        taskID = taskID+1
        createTimer(now.plusMinutes(30)) [|
            if(puertaFrigorificoAbierta == false){
                sendCommand(RepetirAvisoComida, ON)
            }
            ]
end
```

Figura 40. Código ejemplo recordatorio de comer.

- El control del número de veces que se realizan algunas acciones pueden tener interés médico, por ejemplo, el número de veces que el usuario hace uso del baño. Estos datos pueden ser comunicados a un familiar para que tenga conocimiento (figuras 41, 42, 43, 44 y45).

```
Contact ContactoInodoro
```

Figura 41. Declaración ítem.

```
var int vecesBano
```

Figura 42. Declaración variable.

```
rule "visita a Baño"
    when
        Item ContactoInodoro changed from 0 to 1
    then
        vecesBaño++
end
```

Figura 43. Código ejemplo contador de visitas al baño.

```
rule "control y reseteo variable baño"
    when
        Time is midnight
    then
    //insertamos el numero minimo que se considera
    //suficiente para avisar para realizar aviso
    if (vecesBano > 20){
        sendMail("mail@gmail.com", "control baño", "mensaje de informacion")
    }
        vecesBano = 0
end
```

Figura 44. Código ejemplo control, reseteo y comunicación.

```
##############################################################################################
#####                          Action configurations                                     #####
##############################################################################################

######################### Mail Action configuration ###################################
#
# The SMTP server hostname, e.g. "smtp.gmail.com"
mail:hostname=smtp.gmail.com

# the SMTP port to use (optional, defaults to 25 (resp. 587 for TLS))
mail:port=587

# the username and password if the SMTP server requires authentication
mail:username=mail@gmail.com
mail:password= password

# The email address to use for sending mails
mail:from=mail@gmail.com

# set to "true", if TLS should be used for the connection
# (optional, defaults to false)
mail:tls=true

# set to "true", if POP before SMTP (another authentication mechanism)
# should be enabled. Username and Password are taken from the above
# configuration (optional, default to false)
#mail:popbeforesmtp=

# Character set used to encode message body
# (optional, if not provided platform default is used)
#mail:charset=
```

Figura 45. Configuración archivo *.cfg* para envío de mail a través de Gmail.

- Los datos se le comunicarán a través de correo electrónico, este medio de comunicación podrá ser usado para otras actividades que se considere oportuno comunicar a la familia como, por ejemplo, la salida de casa del usuario (figuras 46 y 47), cuando se avisará a quien se haya indicado además de poder establecer un estado de dispositivos y elementos en la vivienda. Para simplificar la regla el usuario nos podrá indicar la salida del domicilio a través de un interruptor.

```
Group Persianas (Todo)
Rollershutter   PersianaSalon1
Rollershutter   PersianaSalon2
Rollershutter   PersianaCocina
Rollershutter   PersianaHabitacion
Switch Salida_Casa
```

Figura 46. Declaración de ítems.

```
rule "salida de casa"
    when
        Item Salida_Casa changed from OFF to ON
    then
        //IP Power WIFI apaga todo
        mqttOutMessage.sendCommand ("NEW_TASK 123 0 SWITCH_OFF 0")
        mqttOutMessage.sendCommand ("NEW_TASK 123 0 SWITCH_OFF 1")
        mqttOutMessage.sendCommand ("NEW_TASK 123 0 SWITCH_OFF 2")
        mqttOutMessage.sendCommand ("NEW_TASK 123 0 SWITCH_OFF 3")
        Luces?.members.forEach(light|
            postUpdate(light, OFF)
        )
        Persianas?.members.forEach (persiana|
            sendCommand (persiana, DOWN)
        )

        sendMail("mail@gmail.com", "control baño", "muchas veces y numero")
end
```

Figura 47. Código ejemplo control salida de casa.

## 4.3- VISUALIZACIÓN

En este apartado se va a tratar de la herramienta que usamos para la visualización del domicilio en *openHAB*.

Durante el desarrollo del proyecto, que se inició con las versiones de *openHAB* 1.8.2, *openHAB* se ha ido actualizando hasta la versión 2.0, en la cual, aunque el modo de funcionamiento a vista de usuario es totalmente igual, desde el punto de vista interno ha sufrido ciertos cambios. Respecto a lo que a nivel usuario se refiere sigue siendo completamente igual en la creación de ítems, elaboración de reglas y creación del '*sitemaps*'.

Debido al interés en realizar en el '*sitemaps*' un proyecto con una visualización más atractiva. En la búsqueda de diferentes herramientas para realizar esta parte se ha encontrado *HABmin2* [19], que realmente es otra interfaz gráfica para *openHAB* y, que en la versión para openHAB2.0 añade la funcionalidad de *FloorPlanEditor* [20], que será la base de nuestra visualización del domicilio.

Cuando nos introducimos en la interfaz de HABmin2 entramos a una página principal en el que a la izquierda hay un menú de selección, como refleja la figura 48.

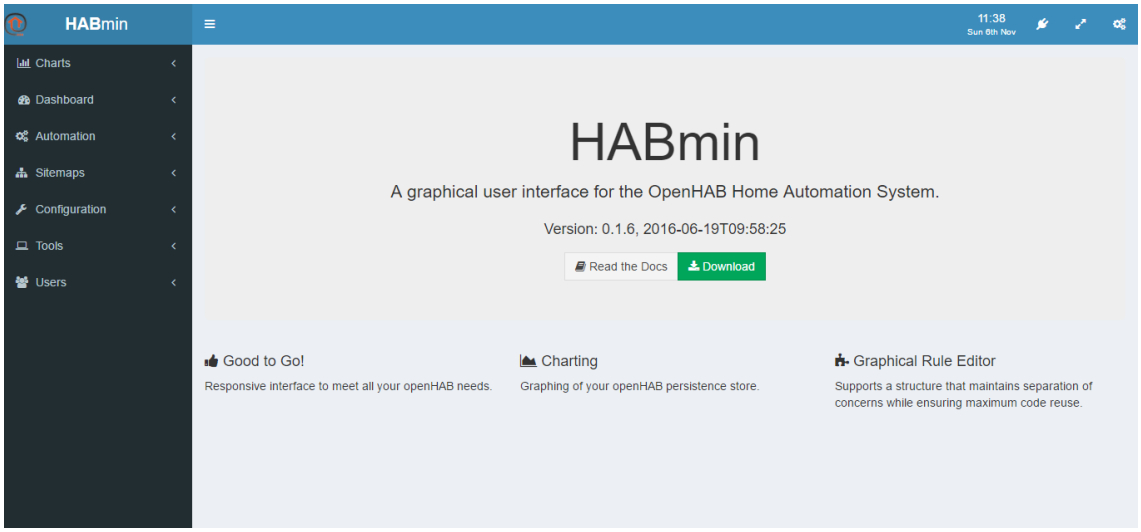



Figura 48. Captura del panel inicial de HABmin2.

Si nos introducimos en el menú *DashBoard* (figura 49), se podrá crear un tablero en el que introducir los elementos que deseemos, como si estuviéramos creando un '*sitemaps*', para poder acceder en una página a todos en conjunto y no tener que visitar apartado por apartado del menú.

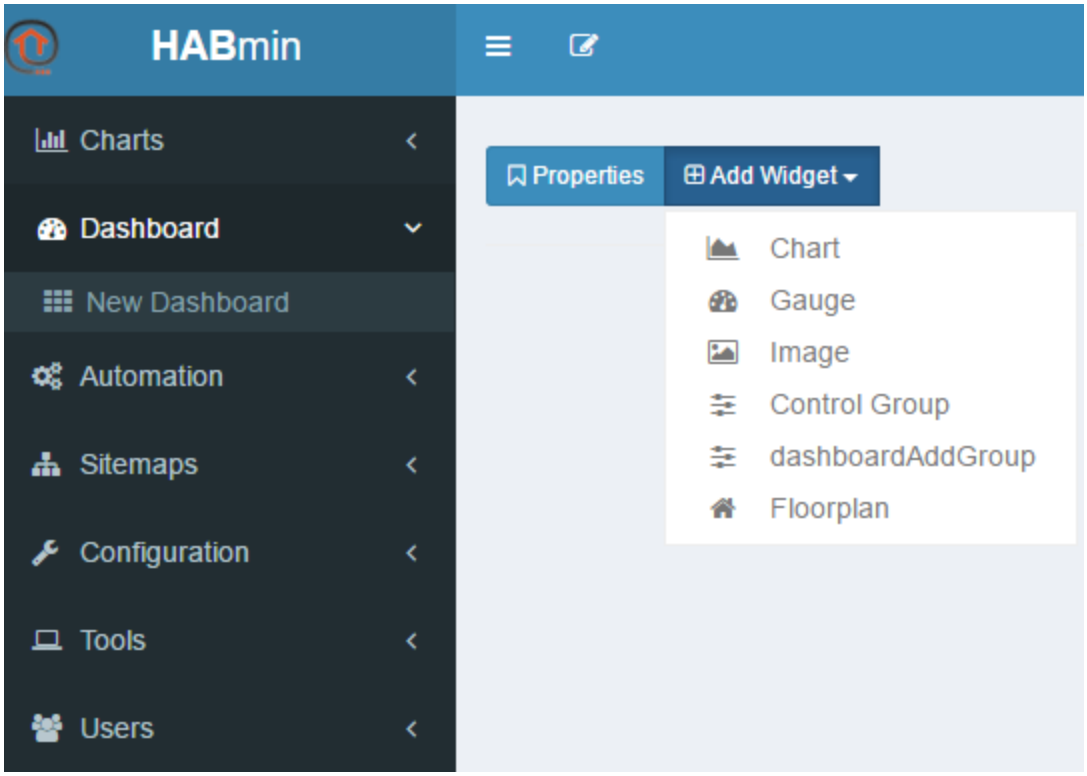



Figura 49. Captura panel *Dashboard* de HABmin2.

En el apartado *Automation/Rules* encontramos el editor de reglas, que en esta interfaz gráfica (figura 50) nos da la opción de crearlas de forma gráfica. Este editor también tiene la opción de crearla a partir de texto (figura 51).

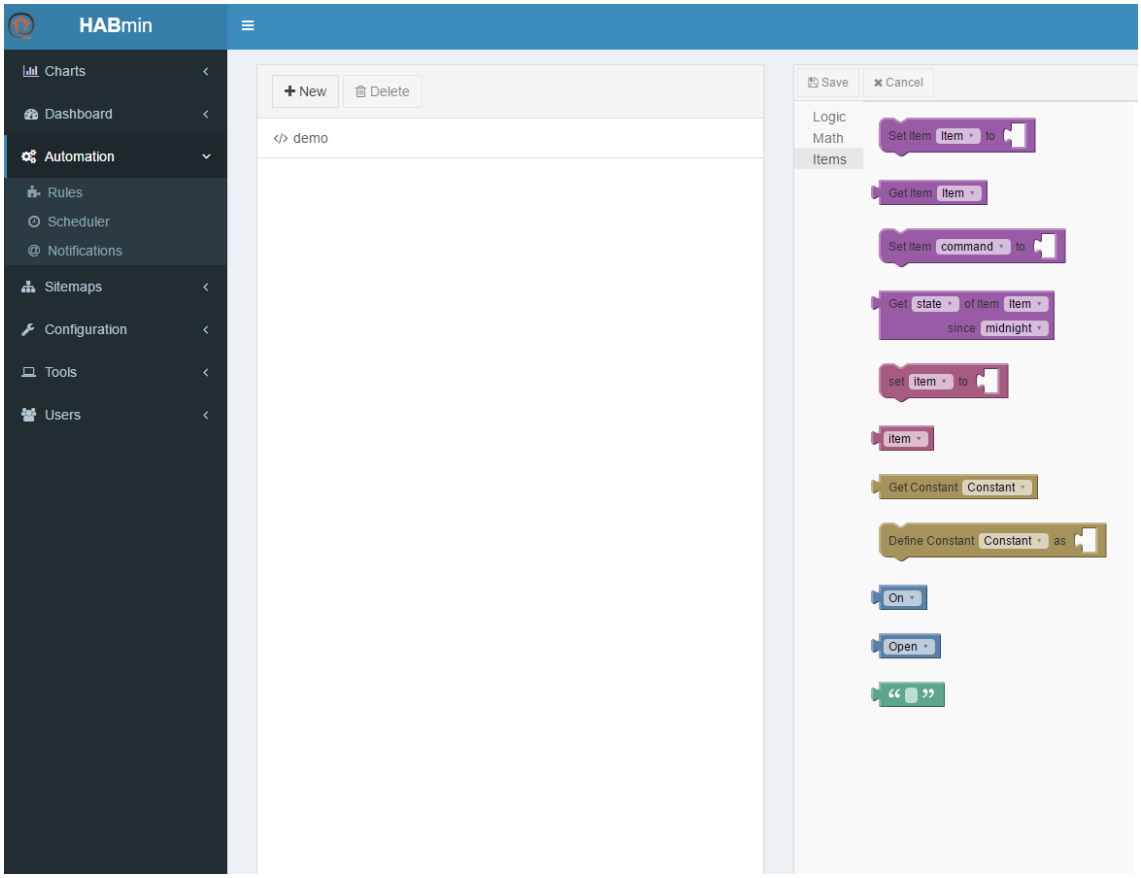



Figura 50. Captura del panel de reglas, versión grafica de HABmin2.

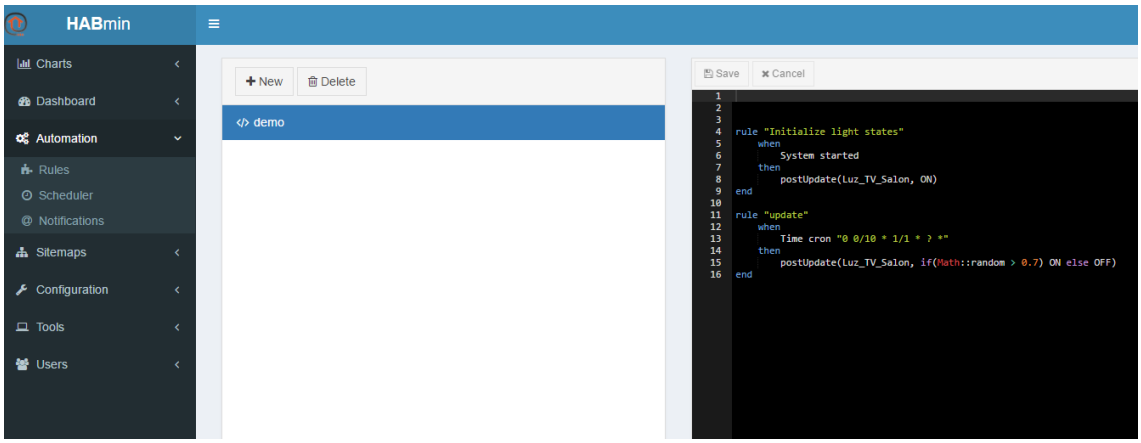



Figura 51. Captura del panel de reglas de HABmin2.

En el apartado de configuración caben destacar dos de los apartados. Primero el apartado de *Items* (figura 52), el cual nos cargará todos los ítems declarados en el

archivo correspondiente del proyecto y a través del cual podremos localizarlos, ver su declaración, tipo, grupo, etiqueta, etc. de una forma más sencilla.

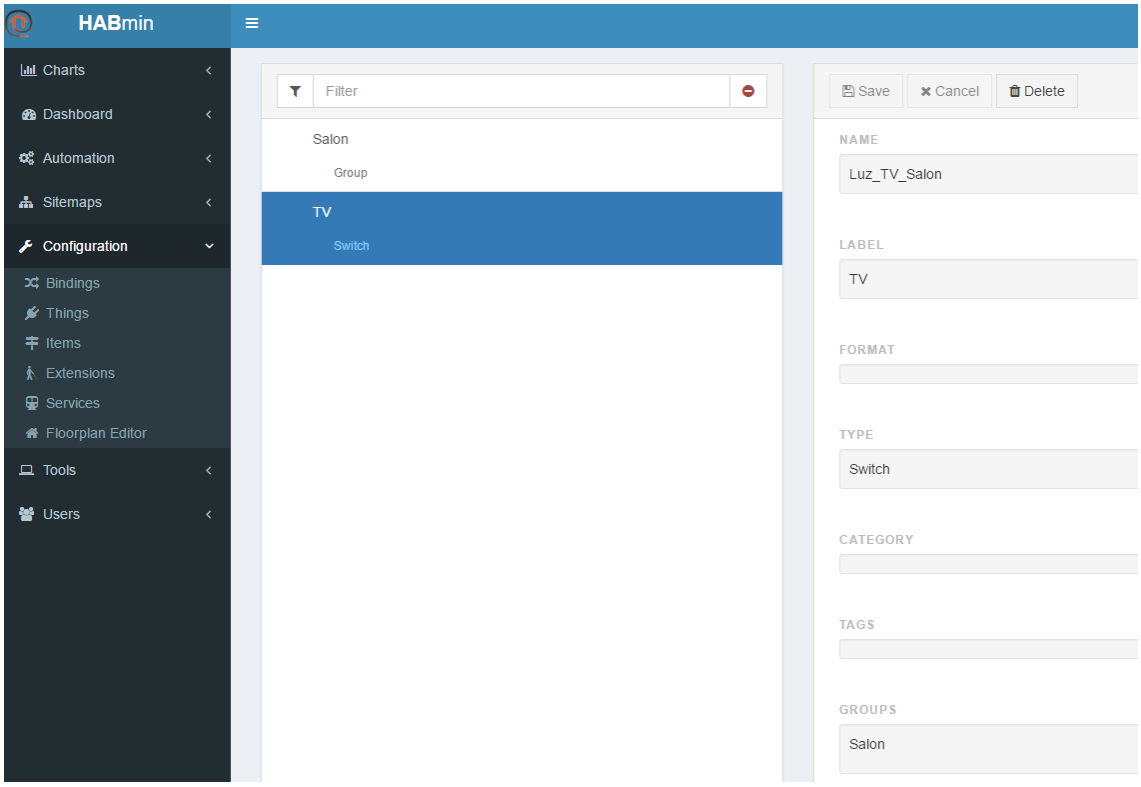



Figura 52. Captura del panel *Configuración/Items* de HABmin2.

Por último, y el aspecto más importante en cuanto a visualización, es la herramienta *FloorPlanEditor*. El uso de la herramienta es tan sencillo como seleccionar una imagen correspondiente con el domicilio al que se quiera referir (un plano de la vivienda en 2D o 3D). Una vez seleccionada se cargará la imagen y, encima de ella, se hará *click* en la zona donde se encuentre el sensor que se vaya a enlazar.

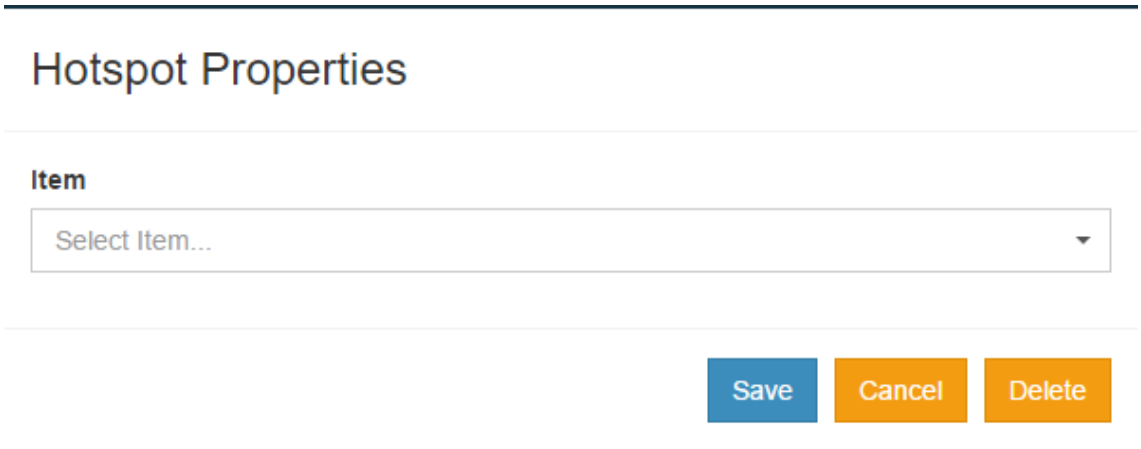



Figura 53. Captura del panel *configuración/Item/selección de ítem* de HABmin2.

Una vez seleccionado el ítem (figura 53) se guardará y aparecerá en el plano un widget que nos informará del estado del ítem (figuras 54 y 55). Si se pulsa en el widget nos dará la información correspondiente al ítem con el que está vinculado.

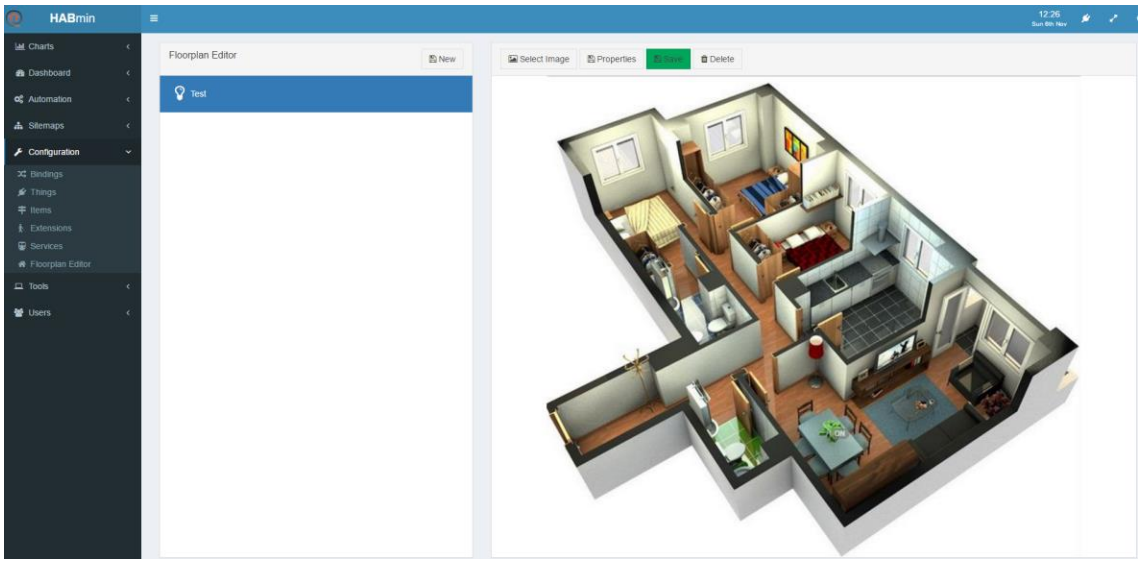



Figura 54. Captura del panel de *configuración/FloorPlanEditor* de *HABmin2*.


Figura 55. Visualización del estado del ítem en *FloorPlanEditor* de *HABmin2*.

# Capítulo 5.

# EXPERIENCIA

Para la conclusión del proyecto se ha realizado una experiencia que contiene el uso de todos los elementos con los que se ha contado para el desarrollo del mismo.

El escenario donde se ha realizado es un laboratorio de dos habitaciones en el que se han dispuesto diferentes sensores como se muestra en la figura 56.

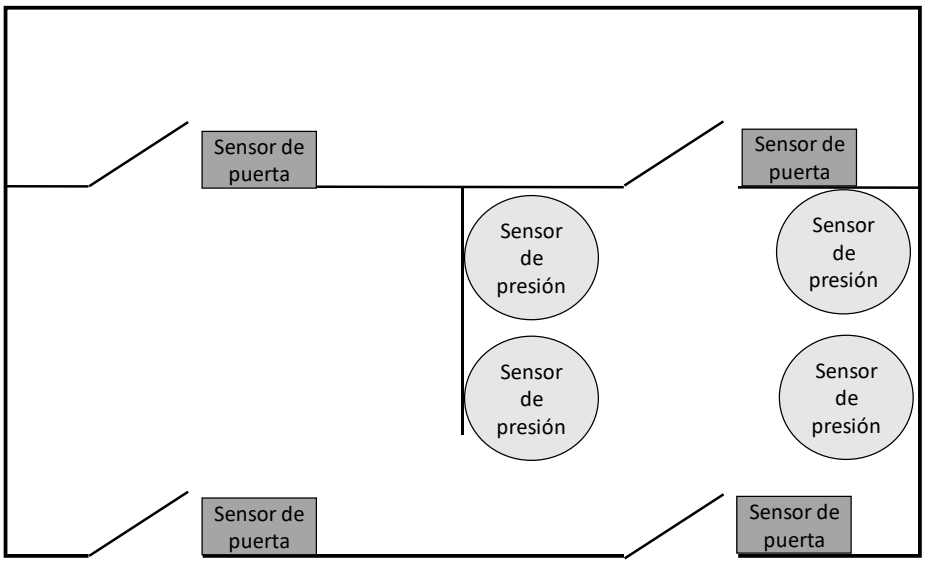



Figura 56. Plano del laboratorio.

Los sensores que se han instalado han sido cuatro sensores de presión (figura 57) para conocer si se encuentra alguien sentado y cuatro sensores de apertura y cierre de puerta (figura 58) para conocer el estado de las mismas.

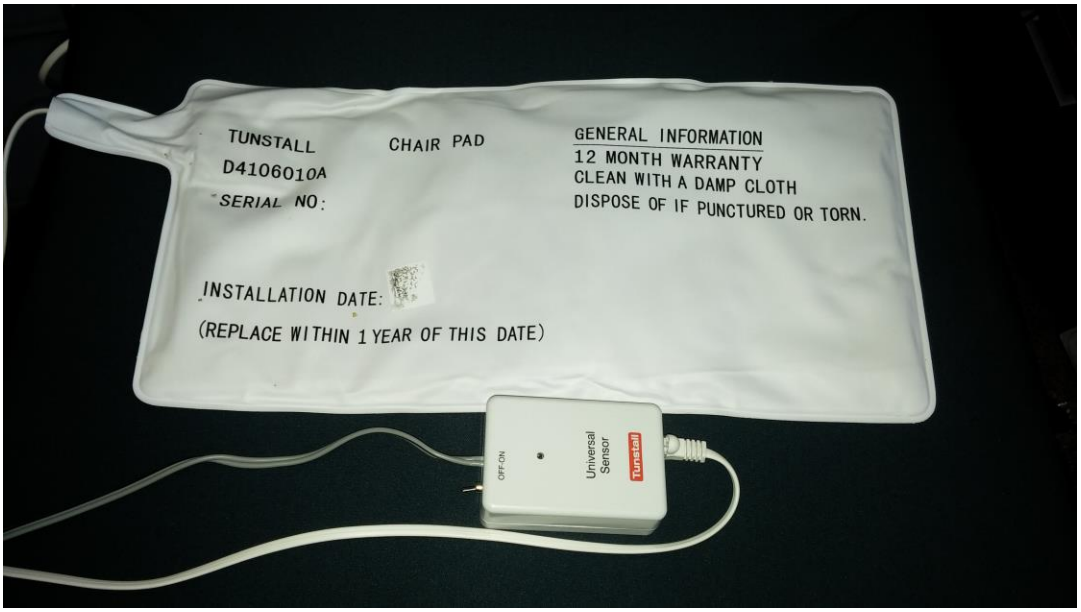



Figura 57. Instalación sensor de presión.

Figura 58. Instalación de sensor de apertura y cierre de puerta.

El esquema de conexiones presentado en el capítulo 4 ha sufrido algunas modificaciones debido a las conexiones de red disponibles en el laboratorio, así como las restricciones de estas redes. Así, el esquema resultante ha sido el que se muestra en la figura 59.

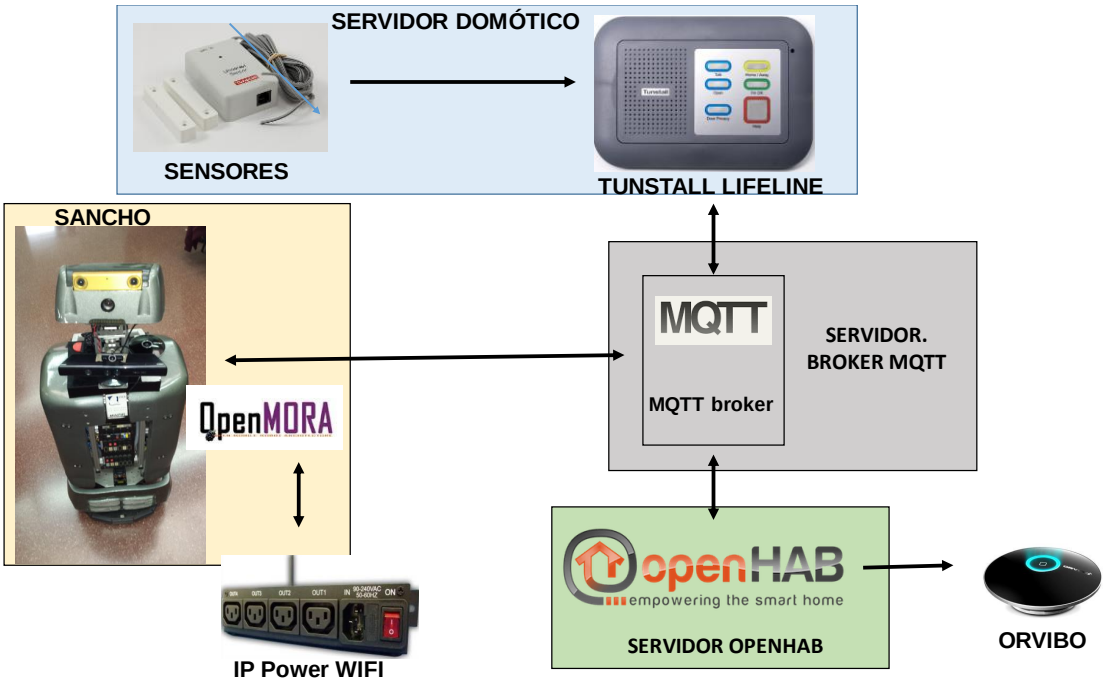



Figura 59. Esquema de conexiones para la experiencia.

Para la realización de las pruebas se ha integrado el dispositivo *Orvibo* al Robot SANCHO, obteniendo así la capacidad de comunicar acciones a los dispositivos eléctricos que hayamos configurado a través de infrarrojos en diferentes habitaciones.

Debido a las comunicaciones disponibles en el entorno del laboratorio y sus restricciones, para la elaboración de la experiencia se ha creado un acceso WIFI a un dispositivo móvil que nos permitirá realizar las comunicaciones.

El caso que se plantea es el control de la ventilación del laboratorio, controlando un ventilador, a través de IP Power WIFI, y el aire acondicionado, situando el robot SANCHO en frente y comunicando la frecuencia adecuada mediante el dispositivo *Orvibo*. Los casos con los que nos encontramos son:

- Que se active un sensor de presión por la entrada de alguna persona, por lo que, si todas las puertas están cerradas se encenderá el aire acondicionado o se reducirá un grado la temperatura en caso de que ya estuviera encendido. Si alguna puerta estuviera abierta se activaría o mantendría el ventilador.
- Que se desactive un sensor de presión por la salida de alguna persona, por lo que se incrementaría un grado la temperatura o, en caso de que fuera la única persona que se encontraba en el laboratorio, se apagaría el aire acondicionado o el ventilador.
- Que se abra una puerta. Si fuera la única puerta abierta se desconectaría el aire acondicionado y se encendería el ventilador.
- Que se cierra una puerta. Si no hubiera ninguna puerta más abierta se apagaría el ventilador y se conectaría el aire.

El código de estos casos de uso se puede ver en las figuras 60, 61, 62, 63 y 64.

```
String mqttSillaA   "Silla A"   {mqtt="<[mosquitto:sancho/amI/tunstall/mapir
                                 /lab236/chair/01:state:default]"}
String mqttSillaB   "Silla B"   {mqtt="<[mosquitto:sancho/amI/tunstall/mapir
                                 /lab236/chair/02:state:default]"}
String mqttSillaC   "Silla C"   {mqtt="<[mosquitto:sancho/amI/tunstall/mapir
                                 /lab236/chair/03:state:default]"}
String mqttSillaD   "Silla D"   {mqtt="<[mosquitto:sancho/amI/tunstall/mapir
                                 /lab236/chair/04:state:default]"}

String mqttPuerta236A   "Puerta 236-A"  {mqtt="<[mosquitto:sancho/amI/tunstall
                                        /mapir/lab236/door/10:state:default]"}
String mqttPuerta236B   "Puerta 236-B"  {mqtt="<[mosquitto:sancho/amI/tunstall
                                        /mapir/lab236/door/11:state:default]"}
String mqttPuerta237A   "Puerta 237-A"  {mqtt="<[mosquitto:sancho/amI/tunstall
                                        /mapir/lab236/door/12:state:default]"}
String mqttPuerta237B   "Puerta 237-B"  {mqtt="<[mosquitto:sancho/amI/tunstall
                                        /mapir/lab236/door/13:state:default]"}

String mqttOutMessage       {mqtt=">[mosquitto:sancho/Debug:command:*:default]"}
```

Figura 60. Declaración de Items.

```
rule "sale persona"
    when
        Item mqttSillaA changed from 1 to 0 or
        Item mqttSillaB changed from 1 to 0 or
        Item mqttSillaC changed from 1 to 0 or
        Item mqttSillaC changed from 1 to 0
    then
        numPersonas = numPersonas -1
        if (numPersonas == 0){
            //mover robot
            mqttOutMessage.sendCommand ("NEW_TASK " + userID.toString +
                " " + taskID.toString + " MOVE CARLOS"
            )
            taskID = taskID+1
            //apagar aire
            executeCommandLine("python /home/pc/orvibo-master/orvibo/orvibo.py
                 -i 192.168.243.43 -e off.ir")
        }else if(numPuertasAbiertas == 0){
            //mover robot
            mqttOutMessage.sendCommand ("NEW_TASK " + userID.toString +
                " " + taskID.toString + " MOVE CARLOS"
            )
            taskID = taskID+1
            //bajo grado
            executeCommandLine("python /home/pc/orvibo-master/orvibo/orvibo.py
                 -i 192.168.243.43 -e bajar.ir")
        }else{
            //enciendo o mantengo ventilador
            mqttOutMessage.sendCommand ("NEW_TASK 123 0 SWITCH_ON 0")
        }
end
```

Figura 61. Código de la regla "sale persona"

```
var Integer taskID = 0
var Integer userID = 123
var int numPuertasAbiertas = 0
var int numPersonas = 0

rule "entra persona"
    when
        Item mqttSillaA changed from 0 to 1 or
        Item mqttSillaB changed from 0 to 1 or
        Item mqttSillaC changed from 0 to 1 or
        Item mqttSillaD changed from 0 to 1
    then
        numPersonas=numPersonas +1
        if (numPuertasAbiertas > 0){
            //encender o mantener ventilador
            mqttOutMessage.sendCommand ("NEW_TASK 123 0 SWITCH_ON 0")
            taskID = taskID+1
        }else if (numPersonas == 1){
            //mover robot
            mqttOutMessage.sendCommand ("NEW_TASK " + userID.toString +
                " " + taskID.toString + " MOVE CARLOS"
            )
            taskID = taskID+1
            //encender aire
            executeCommandLine("python /home/pc/orvibo/orvibo.py -i
                                192.168.243.43 -e on.ir")
        }else{
            //mover robot
            mqttOutMessage.sendCommand ("NEW_TASK " + userID.toString +
                " " + taskID.toString + " MOVE CARLOS"
            )
            taskID = taskID+1
            //sumar grado
            executeCommandLine("python /home/pc/orvibo/orvibo.py -i
            192.168.43.243 -e subir.ir")
        }
end
```

Figura 62. Declaración de variables y código de la regla "entra persona"

```
rule "se abre puerta"
    when
        Item mqttPuerta236A changed from 0 to 1 or
        Item mqttPuerta236B changed from 0 to 1 or
        Item mqttPuerta237A changed from 0 to 1 or
        Item mqttPuerta237B changed from 0 to 1
    then
        numPuertasAbiertas = numPuertasAbiertas +1
        //encender o mantener ventilador
        mqttOutMessage.sendCommand ("NEW_TASK 123 0 SWITCH_ON 0")
        if(numPuertasAbiertas == 1){
            //mover robot
            mqttOutMessage.sendCommand ("NEW_TASK " + userID.toString +
                " " + taskID.toString + " MOVE CARLOS"
            )
            taskID = taskID+1
            //apagar aire
            executeCommandLine("python /home/pc/orvibo-master/orvibo/orvibo.py
                 -i 192.168.243.43 -e off.ir")
        }
end
```

Figura 63. Código de la regla "se abre puerta"

```
rule "se cierra puerta"
    when
        Item mqttPuerta236A changed from 1 to 0 or
        Item mqttPuerta236B changed from 1 to 0 or
        Item mqttPuerta237A changed from 1 to 0 or
        Item mqttPuerta237B changed from 1 to 0
    then
        numPuertasAbiertas = numPuertasAbiertas -1
        if(numPuertasAbiertas == 0){
            //apagar ventilador
            mqttOutMessage.sendCommand ("NEW_TASK 123 0 SWITCH_OFF 0")
            //mover robot
            mqttOutMessage.sendCommand ("NEW_TASK " + userID.toString +
                " " + taskID.toString + " MOVE CARLOS"
            )
            taskID = taskID+1
            //encender aire
            executeCommandLine("python /home/pc/orvibo/orvibo.py -i
                192.168.243.43 -e on.ir")
        }
end
```

Figura 64. Código de la regla "se cierra puerta"

# Capítulo 6.

# CONCLUSIONES Y TRABAJOS FUTUROS

Este proyecto se ha centrado en el estudio de la plataforma *openHAB* orientado a su utilización en domicilios habitados por una persona mayor que necesite de cierta ayuda en su quehacer diario y/o a su monitorización continua mediante el seguimiento de sus actividades.

Debido a la mayor esperanza de vida de las personas actualmente y de las dificultades inherentes al envejecimiento con las que nos encontramos, se ha considerado necesario el estudio de plataformas con las que ofrecer facilidades a los diseñadores que implanten sistemas de este tipo.

La elección de *openHAB* como plataforma para la realización de este proyecto se debe a que no es simplemente una herramienta software para la automatización de procesos o para la realización de sistemas domóticos, sino que además proporciona un espacio común para la comunicación de las diferentes tecnologías con las que nos encontramos constantemente en el panorama tecnológico actual.

Las comunicaciones de entrada a *openHAB*, es decir, la comunicación de los estados de los sensores que se han instalado, sigue el protocolo MQTT. Este protocolo tiene como ventajas que las soluciones basadas en él consumen pocos recursos de computación y memoria, que tiene un tiempo de respuesta muy rápido y, además, que presenta un modelo de publicación y suscripción que permite al emisor y receptor estar desacoplado.

Con respecto a las comunicaciones de salida de *openHAB*, además de usar el protocolo MQTT como en las entradas, también se han usado *addons* disponibles en *openHAB*, como por ejemplo el usado para *Orvibo* que nos permite ejecutar en el terminal de un sistema operativo Linux o los usados para enviar correos electrónicos una vez realizada la correspondiente configuración.

El actuador más importante que incorpora el proyecto es el uso de un robot, SANCHO en nuestro caso. La incorporación de un robot dota al sistema de la capacidad de poder situarse en cualquier punto del domicilio para darnos una visión más cercana y real de la situación actual, además de poder dotar al robot con actuadores como *Orvibo*, dando la posibilidad de hacer uso de dispositivos electrónicos que se manejen mediante infrarrojos con una sencilla configuración.

La visualización conseguida a través de la interfaz de usuario *HABmin* ofrecida por *openHAB* a partir de la versión 2.0 ha ofrecido la posibilidad de tener una óptima visualización del hogar en tiempo real dando la posibilidad de conocer todos los aspectos que, una vez configurados con antelación, se quieran exponer en el servidor

web al que accedemos, ofreciendo la posibilidad del estudio de la situación del domicilio en el momento sin estar presente.

La evolución de este proyecto puede ser extensa con respecto a los trabajos futuros que se pueden realizar. Una vez explicado que la creación y evolución de estos sistemas son necesarios debidos a la evolución de la sociedad, solo queda seguir trabajando en ellos para mejorarlos y adaptarlos a nuevas necesidades y usuarios. Aprovechado la capacidad de *openHAB* para la conexión de diferentes tecnologías, la conexión con cualquier tipo de electrodoméstico, no solo aquellos con estados de encendido y apagado sino con electrodomésticos en los que podamos introducir y modificar los diferentes programas que tengan como, por ejemplo, una lavadora.

También es necesaria la evolución de la virtualización del hogar, haciéndola lo más visual posible y, también importante, en tiempo real. Teniendo cada vez más empresas, universidades e instituciones públicas que investigan y ofrecen servicios de teleasistencia, la mejora en la virtualización traerá consigo una mejora en estos servicios que, en vez de estar basados en la comunicación por voz como es actualmente, también puedan basarse en una visión del hogar virtual y conexiones con el sistema instalado en el domicilio.

# Capítulo 7.

# BIBLIOGRAFÍA


[1] Oficina europea de estadística: http://www.ec.ueropa.eu/eurostats

[2] Instituto Nacional de Estadística: http://www.ine.es

[3] Asus ZenBo: https://zenbo.asus.com/

[4] Proyecto Giraffpluss: http://www.giraffplus.eu/?lang=es

[5] *openHAB*: http://www.openHAB.es

[6] Documentación de OenHAB: https://github.com/openhab

[7] Quartz: http://www.quartz-scheduler.org/

[8] Cron Maker: http://www.cronmaker.com

[9] Tunstall Lifeline Vi: http://www.tunstall.co.uk/Uploads/Documents/D5707001A3_Lifeline_Vi_User_Guide_UK.pdf

[10] Tunstall sensor universal/puerta: http://www.tunstall.co.uk/Uploads/Documents/Universal%20sensor%20datasheet.pdf

[11] Tunstall sensor de presión: http://www.tunstallhealthcare.com.au/Uploads/Documents/Bedchair_sensor_solution_sheet_v4_WEB.pdf

[12] *Orvibo*: http://www.orvibo.com.es/

[13] IP Power WIFI: http://www.aviosys.com/downloads/manuals/power/9258wifi.pdf

[14] SANCHO Service Robot: http://mapir.isa.uma.es/mapirwebsite/index.php/robots/15-the-sancho-service-robot

[15] OpenMORA: http://openmora.github.io/#!index.md

[16] MRPT: http://www.mrpt.org/

[17] Mosquitto: http://projects.eclipse.org/projects/technology.mosquitto

[18] Modulo python para *Orvibo*: https://github.com/cherezov/orvibo

[19] HABmin2: https://github.com/openhab/org.openhab.ui.habmin

[20] Floor Plan Editor: https://github.com/openhab/org.openhab.ui.habmin/wiki/Floorplan-Editor

# APÉNDICES

## INSTALACIÓN OPENHAB

Para realizar la instalación de *openHAB* tendremos que dirigirnos a la página web *http://www.openhab.org/getting-started/downloads.html* y descargarnos los paquetes *runtime core*, *addons* y, si queremos realizar una primera prueba, *demo setup*.

Para una mejor visualización y desarrollo de los diferentes ficheros es recomendable también descargarse *openHAB designer* (figura 65).

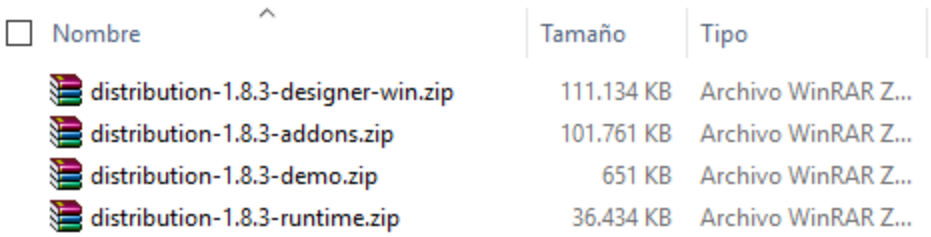



Figura 65. Archivos de descarga para la instalación de *OpenHAB*.

Realizamos la extracción de los ficheros del *zip 'runtime'* y 'demo' en una misma carpeta, en este case la llamaremos *openHAB*. Los otros dos *zip* los descomprimiremos dentro de la carpeta '*openHAB'* pero donde de una nueva carpeta. En el caso de '*addons'* la carpeta la llamaremos '*all_addons'* y en el caso de '*designer'* en una carpeta llamada '*designer'*. Así nos quedará una carpeta (figura 66) llamado *openHAB* que contendrá lo siguiente.

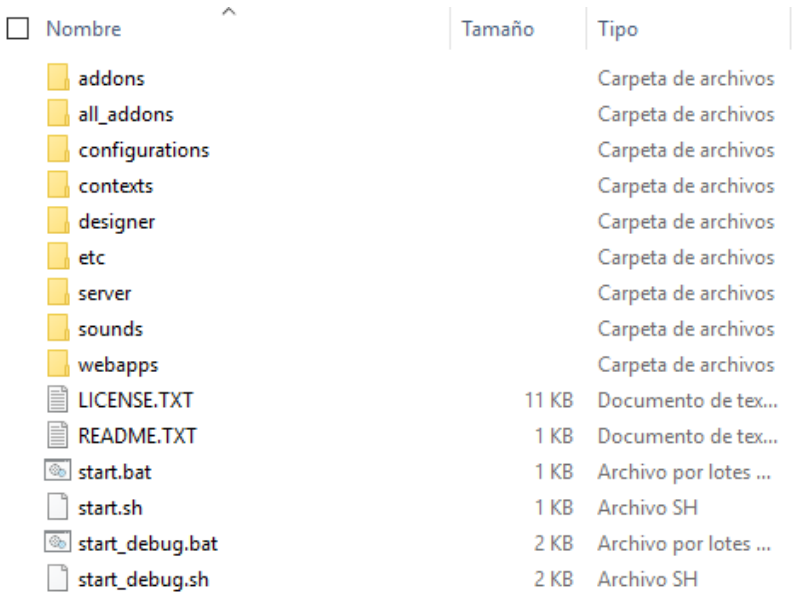



Figura 66. Resultado de la descompresión de los archivos.

Dentro de la carpeta '*configurations'* nos encontraremos con las carpetas '*Items'*, '*rules'* y '*sitemaps'*, en las que dentro encontraremos los archivos que nos aporta *openHAB* de la demo. Estos archivos los podemos modificar para realizar pruebas.

Desde el punto de vista del lanzamiento tenemos dos opciones. La primera opción es lanzar el archivo 'start.bat' (figura 67) y la segunda, para el caso de querer depurar, tenemos la opción de lanzar 'start_debug.bat'. Primero se realizará la carga de los archivos necesarios y después se comenzarán a ejecutar las reglas.

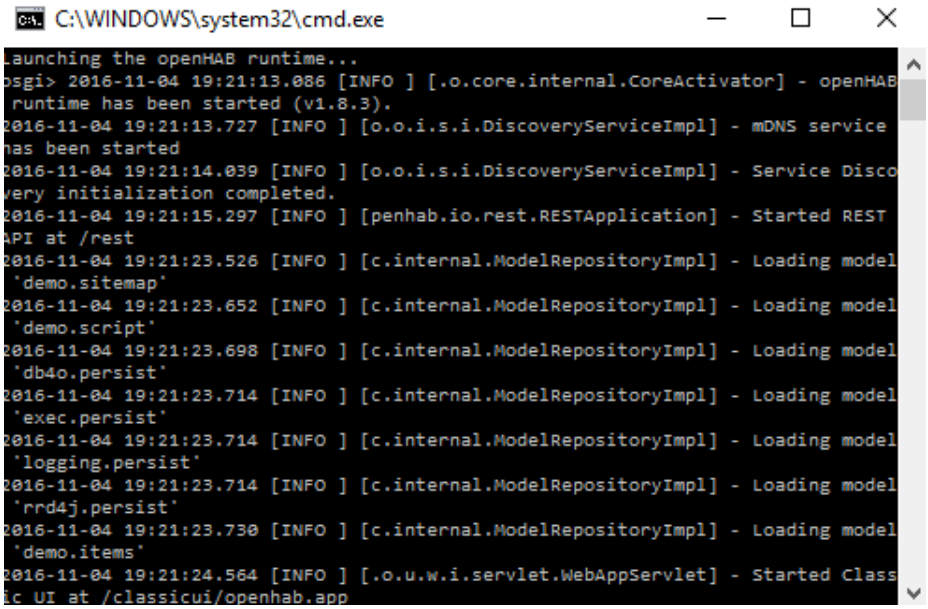


Figura 67. Inicialización de o*penHAB*.

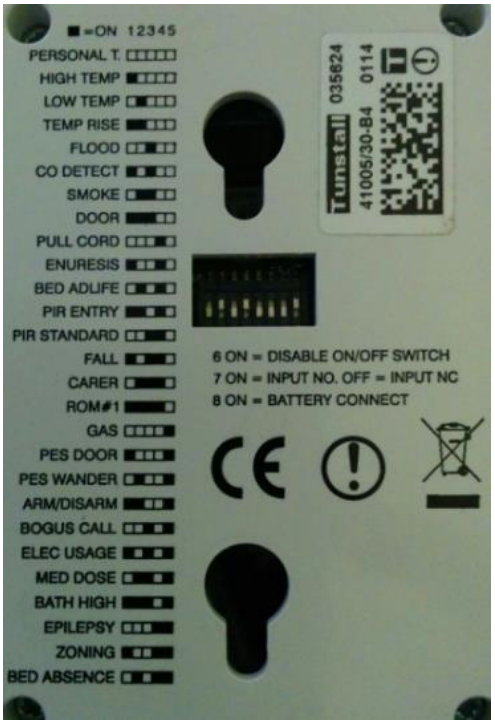
Al introducir en la barra de direcciones del navegador la dirección local y el puerto 8080 se nos mostrará la interfaz de usuario (figura 68) que obtenemos de la demo que nos aportan.


Captura 68. Visualización de la demo en el navegador.

# REGISTRO TUNSTALL

Los sensores de *Tunstall* con funcionalidad *Plug&Play* pueden programarse en la unidad domestica utilizando los siguientes pasos.

Primero debemos disponer los ocho interruptores que tenemos en la parte trasera de la forma que necesitemos, esto será según el sensor que vayamos a conectar al sensor universal y viene indicado en la parte trasera (figura 69) del sensor universal.




Captura 69. Parte trasera del sensor universal.

A partir de aquí debemos tener *Tunstall Lifeline* activo y realizar los siguientes pasos:

Paso 1 - Presione y mantenga presionado el botón de cancelación verde hasta que suene (aproximadamente 5 segundos). La unidad principal anuncia 'Modo de programación' y el botón rojo de alarma parpadea lentamente.

Paso 2 - Presione y mantenga presionado el botón de cancelación verde de nuevo hasta que suene (aproximadamente 3 segundos). Suelte el botón de cancelación, la unidad doméstica anuncia *'Registration Mode'* y el botón rojo de alarma parpadea rápidamente.

Paso 3 - Active el sensor / disparador, la unidad doméstica anunciará el tipo de disparo para confirmar el registro.

Paso 4 - Presione y suelte el botón de cancelación verde. La unidad doméstica emitirá un pitido (el modo de programación saldrá).

Paso 5 - Pruebe el sensor / disparador activándolo y asegurando que eleve una llamada de alarma.