Universidad Nacional de Educación a Distancia

Escuela Técnica Superior de Ingeniería Informática

Proyecto de fin de Grado en Ingeniería Informática

Controladores domóticos inteligentes basados en Arduino

Pelayo Carrasco Montejo

Dirigido por: Fernando López Ostenero

Curso 2021-2022, convocatoria junio

Controladores domóticos inteligentes basados en Arduino

Proyecto de fin de Grado en Ingeniería Informática

de modalidad específica

Realizado por: Pelayo Carrasco Montejo

Dirigido por: Fernando López Ostenero

Fecha de lectura y defensa: 7 de julio de 2022

A todos los que no quisieron apostar por mí... por darme motivos para persistir, y a esa inmensa minoría que sí:

A María Antonia Montejo, entusiasta, literata, perfeccionista y sobre todo Madre, al Jefe, Don Tomás, a Juan Batanero, allá donde estéis en el cielo de los trenes, me habéis hecho ver que la constancia permite alcanzar cualquier sueño (sólo es cuestión de velocidad), a Aquilaino, por enseñarme que se puede remar contra corriente y triunfar, a Fernando López Ostenero, que no deja de demostrar que la buena docencia (la de verdad) es puramente vocacional; Estoy esperando impaciente tu libro de Cifras y Letras.

Y especialmente a la familia pistrafa... Lorena y Tomás (La Lo y Cosito). Si Vesta y Lares existen ha sido gracias a las mil aventuras que os ha tocado vivir, especialmente aquel día de invierno cuando el agua de la ducha salió helada porque Domus olvidó encender el termo.

Resumen

Dice la Wikipedia que la **Domótica** es el conjunto de sistemas capaces de automatizar una vivienda aportando servicios de gestión energética, seguridad, bienestar y comunicación.

En el año 1882, ochenta y cuatro afortunados clientes en New York disfrutaron en sus casas de las primeras luces eléctricas de la historia. Pocos años después el privilegio, convertido en un derecho fundamental, se extendió a tres cuartas partes de la humanidad.

En el año 2022 abundan los productos comerciales que presumen de domótica al ofrecer algo más parecido a un control remoto, un asistente de voz, un temporizador (más o menos complejo) o unos motores eléctricos que mueven persianas y puertas de garaje. La auténtica integración domótica ya está consolidada, pero sólo está al alcance de una élite.

Este trabajo pretende recuperar el concepto de domótica estableciendo unas nuevas bases desde las que se puede levantar una estructura más acorde al fin que se persigue, que es el de dotar de inteligencia a cualquier hogar para mejorar la vida en casa y extender el beneficio a cualquier persona, independientemente de su formación intelectual y presupuesto.

Abstract

Wikipedia says that **Domotics** is that set of systems capable of automating a home by providing energy management, security, comfort and communication services.

In 1882, eighty-four lucky customers in New York city enjoyed the first electric lights in history in their homes. A few years later, this privilege, upgraded into a fundamental right, was extended to three quarters of humanity.

Nowadays (year 2022), there are many commercial products promising home automation by offering something more similar to a remote control, a voice assistant, a timer (more or less complex) or even electric motors moving shutters or garage doors. Authentic home automation integration is already consolidated, but it is only available for a little elite.

This study work aims to recover the concept of home automation by establishing new bases from which a structure more in line with the goal pursued can be built: To provide intelligence to any home, to improve life at home and to extend the benefit to any person, regardless of their intellectual background and budget.

Índice

Índice de tablas

Índice de figuras

Capítulo 1

Introducción

Domótica es *todo conjunto de sistemas capaces de automatizar una vivienda aportando servicios de gestión, seguridad, bienestar y comunicación.*

Adentrándose más en el concepto, existen realizaciones menos publicitadas y más próximas al auténtico objetivo de automatización y gestión de un edificio. Estos productos requieren presupuestos más abultados, personal técnico cualificado y una instalación eléctrica específica, muy diferente a la que se podría encontrar en cualquier domicilio particular.

Figura 1.1: Integración de controladores domóticos con electrodomésticos e iluminación

Mientras tanto la comunidad de desarrolladores crece sin parar extendiendo su talento en forma de creatividad y solvencia técnica. Libres de las ataduras que imponen los plazos y el presupuesto, la capacidad de la comunidad no tiene nada que envidiar al capital humano adscrito a las empresas tecnológicas involucradas en asuntos domóticos. Muchos miembros de la comunidad

de desarrollo libre son también usuarios potenciales de los hogares inteligentes que asumirían gustosamente la carga del desarrollo si tuvieran los medios de contribuir. De hecho no paran de intentarlo practicando ingeniería inversa a los desarrollos comerciales. Las grandes marcas detrás del negocio de la domótica ven en ellos una fuente de competencia potencial que les hace extremar las precauciones para evitar (inútilmente) que surjan productos mejorados fuera de su control.

Todo esto ocurre mientras florecen las tecnologías accesibles y libres que conforman lo que algunos denominan *el internet de las cosas*. Este término engloba una multitud de aparatos de bajo coste y altas prestaciones con arquitecturas libres, enfocadas a sistemas operativos Linux o incluso carentes de ellos.

Todos los sistemas domóticos se componen de un procesador central, un software de control y una colección de aparatos controladores que transmiten los comandos del procesador a los dispositivos bajo control y leen sus señales entregándoselas. Estableciendo una comparación con el mundo natural, un hogar domótico es un ser vivo capaz de adaptarse al entorno y a sus huéspedes. El cerebro sería el procesador central, las líneas de transmisión harían la función del sistema nervioso y los controladores serían los músculos y los sentidos.

Este trabajo de fin de grado va a proponer una arquitectura domótica alternativa, incidiendo en el software de control y en los aparatos controladores. Se asumirá que el procesador domótico será cualquier máquina capaz de ejecutar código *.NET* y que el canal de comunicación será una red ethernet.

En el momento de redactar este trabajo existen dos tipos o familias de controladores domóticos:

- Controladoras industriales de propósito general.
- Componentes modulares de conocidas marcas o estándares.

Los aparatos de propósito general con certificación de equipos industriales suelen ser robustos y costosos. Para que funcionen correctamente requieren un canal de transmisión exclusivo y dedicado; los sistemas formados por controladores de propósito general son diseñados por un Ingeniero industrial y el software necesario para gestionar cualquier despliegue es costoso y se realiza o configura a medida. Todos estos controladores albergan varios dispositivos y siempre del mismo tipo (entradas binarias, salidas binarias, contadores...), acoplándose de forma modular. La instalación de un equipo de control capaz de gestionar los elementos presentes en cualquier habitación de una casa se puede complicar considerablemente en espacio y en coste.
Los componentes modulares de marcas y estándares domóticos son mucho más sencillos de instalar, están pensados para ocupar los espacios habitualmente pensados para la instalación eléctrica convencional del hogar y muchas veces se suministran con documentación suficiente como para que cualquier entusiasta aficionado sea capaz de reconfigurar su casa. En contrapartida son caros, no admiten componentes de otras marcas o estándares en la misma instalación y las posibilidades de ampliación están limitadas dentro de las especificaciones del sistema.

Es llamativo que a pesar de las numerosas opciones comerciales disponibles en la red, la comunidad *maker* ha optado por realizar pequeñas implementaciones partiendo de cero, o bien ampliaciones de otros sistemas que tampoco eran soluciones aceptables. Tal diversidad produce

dispersión; no existe una tecnología que la comunidad haya asumido como un estándar y en la que esté dispuesta a emplear sus esfuerzos para expandir el ecosistema de productos.
El usuario sigue confuso ya que la solución más cercana a la domótica resulta prohibitiva y el márketing le ha hecho creer que adquiriendo un par de bombillas controladas por un asistente de voz, instalar una cámara IP en la puerta de entrada o motorizar las persianas de su salón ya es suficiente para considerar a su hogar como *inteligente*.

Este proyecto va a proponer una nueva solución adoptando los puntos fuertes del mundo industrial y combinándolos con las posibilidades que tiene el desarrollo libre en manos de una comunidad anónima y prometedoramente abundante de entusiastas.

1.1. Objetivos

Las diferentes aproximaciones a la domótica son incompletas. El sistema que debería controlar las viviendas inteligentes tiene que ser asequible, robusto y tiene que tener una arquitectura de código que sea libre y no dependa de un fabricante o un distribuidor. Sólo de esta forma se podrá involucrar a la comunidad en futuros desarrollos. Ni el mercado, ni los desarrolladores han llegado a concebir un sistema que resuelva específicamente las complejidades de la gestión domótica.
Es necesario crear un tipo de controlador suficientemente simple como para que sea sencillo de usar y lo bastante robusto para gestionar correctamente las situaciones degradadas que se dan habitualmente en el mundo real.

Sentar las bases de un nuevo estándar sería una meta muy ambiciosa para un trabajo de fin de grado. Dejar especificada una arquitectura hardware, unas directrices de diseño, unas políticas de codificación y un repositorio de documentación sin haber consensuado las decisiones con un equipo nutrido de expertos sería descabellado. Además habría que definir correctamente cada componente y para ello es necesario realizar pruebas exhaustivas que demuestren que la solución ideada es la opción correcta.
Sin perder de vista el objetivo final, se propondrá el punto de partida que servirá para trazar el resto de la ruta. Un punto de partida que se define así:

- Desarrollar un prototipo de controlador domótico basado en un microcontrolador económico, versátil y suficientemente potente para el trabajo que ha de realizar, capaz de integrar múltiples canales de entrada y salida de información poniendo en contacto sensores y actuadores de un hogar con su correspondiente sistema de gestión.
- Utilizar para ello canales de comunicación fiables, seguros y asequibles.
- Ofrecer múltiples niveles de personalización, desde el básico de un usuario general hasta el de un desarrollador avanzado, sin necesidad de recodificar el software del equipo.
- Garantizar un soporte genérico y eficaz para todos los sensores y actuadores existentes en el mercado y en el futuro.

- Asegurar la escalabilidad del despliegue permitiendo nuevas integraciones futuras sin comprometer las existentes y asegurar un grado de paralelismo máximo aprovechando el potencial de los microcontroladores que gestionan estos dispositivos.
- Desarrollar una plataforma de integración de varios controladores que pueda gestionar la actividad domótica de cualquier tipo de edificio sin ninguna restricción previa.

1.2. Resultados esperados

Al terminar este proyecto se conseguirá un conjunto de máquinas y porciones de código que establecerán una relación fiable y robusta entre los dispositivos físicos bajo control en un edificio y sus representaciones lógicas en una abstracción orientada a objetos. El sistema domótico resultante será tolerante a fallos, contemplará todas las posibles situaciones degradadas (fallo de comunicaciones, averías de dispositivos, fallos de alimentación eléctrica, etc.), será fácil de configurar y cualquier programador de nivel medio podrá escribir código en un lenguaje de propósito general que hará que el edificio adquiera un cierto comportamiento, más sofisticado y potente cuanto más complejo sea este código escrito.

1.3. Consecuencias

Al abstraer la capa física de la lógica, los desarrolladores de código serán capaces de definir el funcionamiento de los edificios inteligentes dando nacimiento a una nueva ciencia domótica, más centrada en los algoritmos que en la disposición de los componentes de hardware. Con el funcionamiento de esta nueva tecnología aparecerán tres nuevos perfiles de profesionales domóticos:

- Desarrolladores de dispositivos: Que crearán código de las tarjetas controladoras capaz de gestionar nuevos tipos de dispositivos físicos. Serán el equivalente domótico a los desarrolladores de drivers para los Sistemas Operativos, que expanden las posibilidades de gestión del kernel con nuevos elementos o mejorando los existentes.
- Integradores de Domótica: Técnicos de cualificación media o baja tendrán a su disposición un nuevo tipo de hardware muy asequible y fácil de mantener, que podrán usar en sus instalaciones futuras extendiendo el número de hogares con domótica.
- Desarrolladores de planes domóticos: Serán programadores de alto nivel capaces de trabajar con abstracciones de un edificio susceptibles de ser simuladas, probando su código o desarrollando librerías de funciones que podrán competir libremente por dotar al mercado de un nuevo tipo de asistente domótico involucrando una vasta colección de dispositivos de todo tipo que trabajarán coordinadamente para satisfacer las necesidades de habitabilidad del cliente.

Como consecuencia la domótica pasará de ser una curiosidad frívola, un costoso sistema a medida en una casa de alto standing o un simple mando a distancia gobernado por voz para

encender unas lámparas a convertirse en una ciencia abierta a la comunidad de programadores sin que tengan que ceñirse a las restricciones de una determinada arquitectura.

Capítulo 2

Estado del arte

En este capítulo se ofrecerá una visión general del mundo de la gestión inteligente de edificaciones en el momento en que se publica este trabajo. Al combinar tantas aplicaciones y objetivos tras una misma palabra, se ha creído interesante comentar lo que recoge la Wikipedia al respecto.

2.1. Objetivos de la Domótica

Manteniendo el símil entre una vivienda inteligente y un ser vivo, son cinco las metas que podrán guiar la gestión coordinada de todos los dispositivos del hogar.

1. **Ahorro energético**

   Una vivienda es un contenedor instalado en el planeta que tiene la misión de cobijar vida humana. Este cobijo puede ser activo y pasivo; el pasivo son todas las estructuras que ofrecen una barrera contra las inclemencias del tiempo, el ruido o la claridad, creando un espacio libre de humedad, frío o calor. El activo se compone de aparatos capaces de mejorar las prestaciones del cobijo pasivo e involucran un cierto consumo de energía. Se podría pensar que la calefacción, la ventilación o el aire acondicionado son creaciones recientes y sin embargo el hombre ya usa fuentes de energía en el hogar desde que inventó el fuego y lo usó para calentarse o cocinar.

   Desde la primera crisis del petróleo en los años 1970, la humanidad tomó consciencia de la escasez de la energía y se fijó metas para minimizar el consumo energético. En los hogares se consiguen grandes avances pasivos con el uso de recubrimientos aislantes y más recientemente, activos gracias a la gestión inteligente de los recursos que consumen energía. Resulta que en este campo la gestión conjunta de múltiples dispositivos que proporciona la domótica permite alcanzar grandes cotas de ahorro con esfuerzos relativamente pequeños. El control de las persianas de una casa permite elevar de forma natural la temperatura de su interior abriendo aquellas que dan al lugar en que hay sol con un consumo despreciable de energía y facilitar el trabajo del aire acondicionado cuando se persigue el fin opuesto si se cierran en el momento en que hay más sol.

   Últimamente se ha impuesto el sistema de subasta del precio de la electricidad dependiendo de los costes de generación. Sumado con el galimatías de diferentes tarifas, costes de peaje e impuestos que ha venido con la instalación de contadores inteligentes, será necesario que el hogar sea capaz de lidiar con esta ensalada de cifras para encontrar los momentos óptimos del día para conectar aquellos consumos que puedan esperar en las horas más económicas. Además, cada vez más hogares cuentan con sus propias formas de generación de energía. El llamado *autoconsumo* ha pasado de ser una tendencia minori-

taria a convertirse en la mejor opción para reducir costes energéticos o incluso acumular pequeñas ganancias volcando a la red los excedentes. En este caso las viviendas son como pequeñas poblaciones y la generación de energía se topa con el mismo inconveniente, que es la acumulación de lo que no se puede consumir en el momento. Hasta ahora los excesos se guardan en paquetes de baterías, costosas y de vida limitada, o se volcaban a la red eléctrica repercutiendo un beneficio muy inferior al del coste real de esa misma energía cuando se recibe de la comercializadora. Sin embargo la domótica podría redirigir el consumo actual minuto a minuto como hacen las centrales eléctricas a gran escala, aprovechando al máximo la potencia generada y reduciendo considerablemente la factura de la luz.

Si la domótica gestionase también la producción y consumo del agua mediante aljibes, pozos y bombas, el hogar podría hacer sus propios acopios hídricos en momentos de lluvia para su uso en períodos de escasez.

2. **Seguridad**

   La seguridad del hogar persigue la protección contra intrusiones y también la detección y mitigación temprana de catástrofes domésticas.

   Los cerrojos robustos y las puertas acorazadas han perdido gran parte de su poder de disuasión. Los delincuentes conocen múltiples técnicas para forzar las protecciones pasivas, de la misma forma que son capaces de burlar los mecanismos de las cajas fuertes de los bancos. El sistema legal es lento en muchos países y la situación de escasez que ha provocado las últimas crisis económicas hayan disparado el fenómeno de la *okupación*, llegando a producirse situaciones que no hace mucho serían consideradas como surrealistas. En cualquier caso ha surgido la necesidad de incrementar la protección de una forma más efectiva y la domótica ofrece grandes soluciones. De la misma forma que un ser vivo puede reaccionar ante los ataques de un entorno hostil, una vivienda inteligente capaz de controlar todos los aparatos y responder a todos los sensores que contiene podrá detectar cualquier patrón de intrusión y responder a él impidiendo o dificultando en gran medida el esfuerzo del atacante hasta que el dueño pueda movilizar a la policía o recupere el control de la situación.

   El control de accesos está relacionado con este tipo de seguridad. Una vivienda inteligente debe ser capaz de discernir qué seres humanos debe admitir y cuáles se tienen que considerar intrusos. La detección puede ser tan simple como un juego de claves privadas de usuario o tan complejo como un sistema de detección biométrica. Las posibilidades se multiplican dividiendo la casa en zonas o validando permisos sólo en momentos concretos del día o del año, de forma que el hogar sabrá en todo momento quiénes están dentro y qué privilegios tienen.

   Igualmente importante para un hogar es que pueda detectar ciertas amenazas domésticas (incendio, inundación, escapes de gas...) en sus primeras fases, alertando a sus habitantes y reaccionando con los medios a su disposición para mitigar en lo posible el problema. Los detectores de humo, de agua, de calor, etc. son baratos, son fáciles de integrar y tienen una fiabilidad garantizada.
   Garantizar una buena seguridad no sólo es conveniente por los problemas que evita, sino porque también supone un considerable ahorro económico. Desde los gastos que se evitan y el tiempo que se deja de perder con las catástrofes hasta el abaratamiento en las primas que las compañías de seguros ofrecen a los hogares que son capaces de autoprotegerse.

3. **Accesibilidad**

   La vivienda inteligente debe ser capaz de ofrecer protección a quienes más la necesitan. Se tiende a relacionar la accesibilidad con la atención a los discapacitados y, aunque se trata de algo importante, en el caso de la domótica, se refiere más bien a la gestión automática de un sistema muy complejo ofreciendo una imagen amigable. De nuevo la metáfora de los seres vivos es perfecta; se puede comprobar cada vez que se acaricia un gato, que proyecta una imagen de cercanía y confianza muy alejada de los complejos mecanismos internos que su organismo debe realizar para mostrarse tan simple. Tras la piel del animal se producen fenómenos muy complejos como la digestión, el funcionamiento de un complicado sistema nervioso, un sistema circulatorio, etc. Todas estas funciones pasan a un segundo plano cuando se piensa en lo que significa tener una mascota. La vivienda domótica debe ser compleja, requiere la instalación de un gran número de sensores, aparatos, luces, motores, interruptores y electroválvulas para que todos ellos trabajen en armonía y ofrezcan una imagen de sencillez máxima.

   Una buena instalación domótica debe pasar desapercibida para el habitante. La única diferencia que debe notar con respecto a una vivienda *tradicional* es que no necesita hacer prácticamente nada para desarrollar su vida en ese hogar. La domótica no se debe ver; sólo percibir, desde el café recién hecho al levantarse al radiador encendido en el momento de despertar, una temperatura agradable a todas las horas del día, una gran sensación de seguridad y una considerable bajada en las facturas de la luz, el agua y el gas.

4. **Comunicaciones**

   Los seres vivos están en contacto permanente con otros seres vivos y con su entorno. Cuanto más avanzada es una civilización, más relacionados están todos sus miembros. El hogar inteligente debe ser capaz de interactuar con sus dueños, con el servicio técnico y con los proveedores de bienes y servicios habilitados. Dentro de los beneficios de la accesibilidad, esa sensación de cercanía y seguridad se tiene que percibir desde cualquier esquina del mundo, ya sea para observar el interior del hogar, ya para encender la calefacción, para avisar de la llegada de un hijo o incluso informar de que todo va bien en un momento determinado. No puede haber límites; valen las redes sociales, el correo electrónico, un navegador, una aplicación de mensajería instantánea o cualquier otra forma de expresión presente o futura. La casa está ahí, debe ser un miembro más del hogar, con un rol activo, capaz de informar de la temperatura interior, pero también de prepararse para el clima venidero consultando la previsión meteorológica, haciendo un pedido a una tienda online de un suministro necesario antes de que se agoten las existencias o ¿por qué no?, avisando al servicio técnico de una probable avería para recibir mantenimiento o reparación antes de que el propio dueño se de cuenta.

5. **Confort**

   Pero por encima de todo, el principal objetivo de una vivienda inteligente es ofrecer a sus habitantes la comodidad del mejor hotel del mundo o de una gran mansión repleta de personal de servicio atento a sus moradores las 24 horas del día. Este confort se percibe en oposición al ambiente hostil del exterior ofreciendo el remanso de paz y seguridad que permite descansar o desconectar del día a día. Todos esos aparatos conectados deben cooperar para ofrecer una experiencia totalmente diferente a la de una vivienda convencional, desdibujando las funciones propias de los elementos del hogar si es preciso con la premisa de ofrecer la máxima atención. Un simple interruptor de pared para encender la

luz del salón debería reconvertirse en un instrumento para informar a la casa de cambios de parecer eventuales. Los humanos son seres de hábitos y si es costumbre cenar en el salón a las ocho y media, a esa hora la luz tiene que estar encendida. Una buena vivienda domótica debe conocer la presencia de sus habitantes, saber quiénes son y deducir a qué hora exacta van a cenar para que encuentren esa luz encendida sin que nadie tenga que volver a tocar la tecla.

El ejemplo perfecto es el de las zonas comunes de cualquier hotel. Ahí están, a disposición de los huéspedes, con la temperatura, la iluminación y hasta la música apropiada, esperando a que se dejen agasajar por el local. La gran diferencia que define a una vivienda inteligente es que será capaz de proporcionar todas estas atenciones de la manera más eficiente y sostenible con el medio ambiente, así que el disfrute saldrá también más económico.

2.2. Arquitecturas domóticas

Todos los sistemas domóticos entran dentro de alguna de estas tres distribuciones de equipos:

1. **Arquitectura centralizada**

   Una máquina denominada **controlador centralizado** recibe información de una red de sensores, la procesa y genera órdenes que dirige a una red de actuadores. La potencia del controlador centralizado será el indicador de la eficiencia del sistema y también es su Talón de Aquiles. Si falla el controlador centralizado, falla todo el sistema. Los sistemas **Scada** se basan en esta arquitectura y se aplican a entornos industriales. Las arquitecturas centralizadas se encuentran en viviendas de alto standing que pueden contar con sus propias habitaciones de control, similares a un centro de cálculo, pero no en hogares corrientes. Es habitual también que la red de comunicaciones tenga canales reservados para la domótica, independientes del resto de actividades de comunicación del edificio.

2. **Arquitectura distribuida**

   Diferentes dispositivos conectados en bus reaccionan a eventos y desencadenan acciones. Puede existir un controlador centralizado, aunque se considera un elemento más del sistema que, aunque dejase de operar, no eliminará la funcionalidad completa. Se han definido protocolos específicos para domótica, como *inBus*,*C-Bus*,*Insteon* o *KNX*, aunque también se podrían utilizar buses y protocolos de automoción como *CAN-Bus* o incluso canales de transmisión en red genéricos como el *RS-486*. Dependiendo de la complejidad del sistema, estos canales pueden ser dedicados (Insteon, KNX, CAN-Bus, RS-486), o funcionar utilizando la propia red eléctrica del hogar (inBus, C-Bus) inyectando y detectando sus señales en la alimentación a 220V. En este último caso se tratará de implantaciones parciales de bajo costo, muy de moda en el momento de realizar este trabajo.

3. **Arquitectura mixta**

   Es similar a las arquitecturas centralizadas, solo que el canal de comunicación es el aire (radiofrecuencia). Los dispositivos emiten y reciben su información de forma abierta o estableciendo un protocolo de encriptación similar al de una red Wifi, pudiendo utilizar este tipo de redes, bluetooth o protocolos especialmente dedicados para estos fines como

ZigBee. Normalmente se utilizan en implantaciones superficiales de bajo costo igual que las redes domóticas basadas en la corriente alterna de alimentación. Debido a la vulnerabilidad del medio, las arquitecturas mixtas se configuran en servicios accesorios más destinados al confort percibido que a la gestión intensiva de los recursos de un hogar.

2.3. Elementos de una instalación domótica

En una instalación domótica podemos distinguir cuatro elementos principales:

1. **Central de gestión o controlador centralizado**

   Es el cerebro del sistema. En instalaciones profesionales es un procesador de grado industrial que alberga un sistema operativo en tiempo real, que puede ser complementado con un equipo dedicado a comunicaciones externas (servidor WEB o multiprotocolo) y otro a gestionar el bus de transmisión.

   En hogares más modestos un computador personal con un sistema operativo estándar como Linux o Windows puede realizar estas funciones si el entorno no es crítico.

   Los sistemas domóticos *ligeros* basados en el asistente de voz pueden incluso delegar las funciones del controlador centralizado en *la nube*, mediante servicios como los del propio asistente (Google Home [1], Amazon - Alexa [2] o ITTT [3]), asumiendo todas las consecuencias de depender de un proveedor externo y que si falla la conexión con internet la domótica dejará de funcionar.

2. **Sensores / Detectores**

   El despliegue de medios es abrumador. El mercado está inundado con sensores capaces de medir absolutamente todo lo relativo a un edificio, desde la temperatura y la humedad a la proporción de un gas determinado en el aire. La mayoría eran imprecisos y tenían un coste prohibitivo antes de los años 1990, pero la situación cambió drásticamente. La dificultad relativa a los sensores consiste en discriminar la información relevante para la gestión domótica entre la increíble cantidad y calidad de los datos adquiridos. La calidad de estos datos se multiplica cuando se combinan las informaciones de sensores de distintos tipos. Un lector RFID [4] puede informar al sistema sobre quién es la persona que está pasando por un determinado lugar de la casa y una cámara acoplada a un sistema de reconocimiento facial puede incluso mejorar esta información. En la metáfora del ser vivo, se puede reconocer la presencia de una persona o una situación de peligro combinando olores, sonidos, imágenes, sensaciones táctiles e incluso sabores del medio exterior. A veces basta una imagen, otras se deduce por el resto de canales sensoriales. Una vivienda bien configurada debe percibir correctamente el entorno y la situación interior combinando todo lo que los sensores pueden informar.

[1] Google Home: https://home.google.com/
[2] Amazon Alexa: https://developer.amazon.com/es-ES/alexa
[3] If This Then That: https://ifttt.com/
[4] Identificacion por radiofrecuencia. Es un sistema de almacenamiento y recuperación de datos remotos que usa dispositivos denominados etiquetas, tarjetas o transpondedores https://es.wikipedia.org/wiki/RFID

3. **Actuadores**

   Son los *músculos* de la casa. Se considera como actuador una luz, un ventilador, la electroválvula que conecta los aspersores del jardín o incluso la cadena musical.

   Cuantos más actuadores tenga el edificio más control podrá ejercer el sistema domótico sobre el medio. Un control parcial apenas permitirá ejercer las funciones típicas de un simple mando a distancia, pero el control pleno hará que el hogar trabaje coordinadamente ofreciendo el fin pretendido sin que importen los medios empleados. Para comprender este concepto basta pensar en el salón en que el habitante de la casa suele leer el periódico. Un control parcial podría encargarse de encender la luz más cercana al lugar de lectura ofreciendo una solución incompleta y limitada. Teniendo en cuenta que el fin que se persigue es obtener una cantidad de luz determinada (cuantificable) para leer el periódico, con suficientes actuadores el sistema domótico puede desarrollar una estrategia para ofrecer el fin perseguido combinando diferentes medios, y es que para leer el periódico de noche puede ser suficiente con encender esa luz, pero a pleno día puede resultar más placentero (y económico) abrir la persiana dejando que entre luz natural hasta que caiga la tarde y entonces encender las luces que hagan falta para ir compensando la falta de radiación solar con el encendido paulatino de luminarias, teniendo en mente que la mejor solución no es sólo la más efectiva sino también la más rentable o sostenible.

   Persiguiendo la accesibilidad de la que se habló antes, sobre todo en viviendas destinadas a personas mayores o niños, es muy interesante que sistemas complejos como la caldera, el aire acondicionado, los radiadores y los sistemas de ventilación sean totalmente autónomos trabajando de forma coordinada para crear la sensación de confort perseguida sin que la persona que se beneficia de ello tenga siquiera que conocer los detalles de los equipos, preferiblemente muy sofisticados y de calidad, bajo control. Si hay que mantenerlos, ya se encargarán los técnicos, porque la vivienda ya les habrá avisado cuando los necesite.

4. **Canales de transmisión**

   La red nerviosa del hogar es imprescindible para comunicar todos los dispositivos entre sí y con la central de gestión. Puede ser dedicada o compartida, cableada o inalámbrica, en bus, daisy chain [5] o en estrella [6]. La inversión depende del grado de integración de la domótica. La fiabilidad del sistema depende estrechamente de la fiabilidad del canal de transmisión. La elección depende mucho del propio inmueble. La normativa de construcción actual establece ya una preinstalación de redes de comunicación pensando en las integraciones domóticas, las viviendas construidas a partir del año 2000 también están preparadas recableando las conducciones telefónicas o de señal de vídeo existentes con los cables de red ethernet, fibra óptica o coaxial metálico que se requiera y establecen ya una caja de empalmes en un lugar relevante concebida para ello.

   El auténtico problema está en las viviendas más antiguas. Replantear las conducciones para albergar las líneas de datos siguiendo la normativa oficial implicaría una costosa obra que frecuentemente supera el coste de la propia actualización domótica. Por ello existen protocolos de transmisión inalámbrica como zigBee, inBus, C-Bus o Z-Wave o alámbrica a través de las líneas de alimentación de corriente alterna como Insteon, KNX o UPB. Con sus posibles problemas y fallos eventuales de transmisión, cualquiera de estos

[5] Topología de red en que los nodos que la componen se encuentran encadenados en círculo formando un corro. Los mensajes de un nodo a otro viajan a través de los nodos intermedios que los separan.

[6] Las estaciones están conectadas directamente a un punto central y todas las comunicaciones se han de hacer necesariamente a través de éste

protocolos permite actualizar inmuebles ofreciendo una solución provisional efectiva hasta la siguiente reforma de importancia.

2.4. Marketing y Domótica

Basta introducir el término en cualquier buscador de la red para conocer la inmensa cantidad de opciones comerciales que invitan al usuario a mejorar su vivienda. Hasta la década de los años 1990 la domótica era una tecnología de vanguardia destinada sobre todo a la ingeniería civil (*Edificios inteligentes*), impensable en viviendas de particulares por la complejidad y el alto coste de los equipos. Con el nacimiento de los microcontroladores de propósito general cobijado por despegue comercial de China y la fuerte consolidación de la informática en el medio doméstico, los precios de la tecnología se hundieron inundando el mercado de productos muy económicos y con grandes prestaciones.

Fue el nacimiento de la llamada comunidad *maker*, el código libre, la difusión del conocimiento en foros especializados y el florecimiento del **DIY** (Do it yourself), el que propició un nuevo campo de afición cultivado por entusiastas con conocimientos universitarios o formación autodidacta, capaces de superar los límites de las prestaciones que ofrecían los productos comerciales, supliendo con ingenio cualquier dificultad inicial. Mientras los proyectos como **Arduino** [7], **Raspberry** [8] o **Microchip PIC** [9] acercaban las complejidades de la informática industrial a la población en general, las grandes empresas tecnológicas se esforzaron por poner en el mercado productos robustos cuyas prestaciones superaban cualquier experiencia previa.

El primer paso fueron los llamados **Asistentes integrados**; software de alto nivel capaz de satisfacer todas las peticiones del usuario expresadas con su lenguaje natural hablado y procesadas por complejos centros de computación en red, en la denominada **nube**, en continuo proceso de desarrollo y mejora, accesibles mediante pequeños aparatos electrónicos que se conectan con internet a través de una conexión inalámbrica dando la ilusión de inmediatez y sencillez ante cualquier pregunta o encargo. Los asistentes se pueden integrar con estructuras virtuales que accionan relés a través de otros pequeños aparatos tras un proceso sencillo de configuración y permiten cierto grado de automatización e inteligencia compatible con lo esperado para un hogar domótico.

Pasada la euforia por la novedad, el alcance logrado no cumple los objetivos de integración que definen un edificio inteligente y se queda en una simple mejora interesante.

2.5. Soluciones domóticas comerciales existentes

El término *domótica* es demasiado general como para definir un producto concreto. A continuación se pondrán ejemplos de varias realizaciones comerciales que llevan este título, expresando

[7] Plataforma electrónica de código abierto basasa en hardware y software fácil de usar. https://www.arduino.cc/

[8] Familia de microcomputadores de propósito general y precio reducido https://www.raspberrypi.org/

[9] Microcontroladores programables para fines industriales y profesionales https://www.microchip.com/

sus fortalezas y también sus debilidades:

2.5.1. Domótica ligera

Son los sistemas de acceso, concebidos para que el cliente haga su primer hogar inteligente.

2.5.1.1. Alexa

Es un asistente virtual desarrollado por Amazon [10], que se empleó por primera vez en el altavoz inteligente *Amazon Echo* [11] mostrado en la figura 2.1. Alexa puede controlar varios dispositivos

Figura 2.1: Altavoz inteligente Echo Dot de Amazon

inteligentes compatibles con el sistema como focos, interfonos, cámaras de videovigilancia e interruptores inteligentes.

Los usuarios pueden extender las habilidades de Alexa instalando **skills**, que son funcionalidades adicionales desarrolladas por terceros, similares a las aplicaciones. Además de los comandos directos se pueden crear rutinas para automatizar los dispositivos inteligentes.

Ventajas: Bajo coste de inicio, facilidad de configuración, tecnología en constante proceso de mejora.

Desventajas: Es un sistema propietario de Amazon, depende de la nube (si falla la conexión a internet o la señal wifi, el sistema entero cae), integrar todos los dispositivos del hogar resulta muy costoso, el canal de transmisión (wifi) es poco fiable. Dependencia técnica del propietario del sistema, que puede alterar el servicio, cobrar por él o anularlo.

[10] Tienda online propiedad de Jeff Bezos https://www.amazon.com/
[11] Altavoz inteligente desarrollado por Amazon para interactuar con su asistente virtual Alexa

2.5.1.2. Google Home

Es una familia de componentes centrados en el hogar que comenzó con el altavoz inteligente desarrollado por Google [12] de la figura 2.2. El primer dispositivo se anunció en mayo de 2016. A traves del asistente Google (*Google assistant*), el usuario interactua con el sistema mediante comandos de voz. Es posible integrar los comandos con un gran número de dispositivos, tanto de la marca como de terceros.

Figura 2.2: Altavoz inteligente Nest de Google (Alphabet)

Ventajas: Las mismas que Alexa

Desventajas: Las mismas que Alexa

2.5.1.3. ITTT

Son las siglas de *If this, then that* [13]. Es un servicio en la nube que permite crear patrones de comportamiento en respuesta a eventos generados en un hogar. Tras dar de alta cada dispositivo en el servidor, el servicio en la nube pasa a capturar los eventos generados desencadenando las acciones oportunas.

Ventajas: Permite recibir eventos y enviar órdenes desde y hacia una gran cantidad de dispositivos y asistentes independientemente de su tecnología y alcance.

[12] Portal de Google Home: https://home.google.com/
[13] https://ifttt.com/

Desventajas: Sólo funciona cuando existe internet, la versión de pago requiere suscripción periódica y la gratuita es demasiado básica. Debido a su forma de funcionar, la actividad de la casa puede quedar expuesta o comprometida a través de la red. Es demasiado lento para algunos comandos y resulta complejo programar las interacciones que no son básicas.

2.5.1.4. Z-wave

Es un estándar internacional [14] de comunicaciones inalámbricas a través de ondas de radio de 433Mhz siguiendo el esquema de la figura 2.3. Requiere un centro de control y uno o varios dispositivos compatibles que pueden ser de diferentes fabricantes. Permite integración con los asistentes de voz más populares y admite scripts denominados *escenas* dentro de las posibilidades concebidas por el fabricante.

Figura 2.3: Esquema promocional de Z-Wave

Ventajas: Es rápido de instalar y de configurar. Tiene un buen factor de escalabilidad. No es necesario contar con asistencia técnica para dar de alta nuevos dispositivos.

Desventajas: Las comunicaciones inalámbricas son susceptibles de error, se generan perturbaciones radioeléctricas y es posible que existan interferencias con otros aparatos en frecuencias próximas. El precio de cada componente es caro. Los tiempos de respuesta son largos. Los scripts o *escenas* son difíciles de crear y no admiten construcciones complejas. La lista de fabricantes adheridos al estándar es pequeña y como sus respectivas producciones son disjuntas no existe un factor competencia que estimule unos precios competitivos o mueva un interés por la calidad.

[14] Web de Z-wave: https://zwave.es/

2.5.1.5. Delta Dore

Se trata de otra aproximación similar a Z-wave, en este caso en manos de una empresa francesa [15] creada en el año 1970, que pasó de fabricar termostatos y reguladores de calefacción a concebir un esquema de transmisión de datos inalámbrico. Después se expandió por Europa y estableció su propia línea de productos para domótica.

Ventajas: Todas las de una red inalámbrica.

Desventajas: Las mismas de Z-wave.

2.5.1.6. Zigbee

Se trata de otro estándar (IEEE 802.15.4) [16] de redes inalámbricas de área personal similar a las WAN, pero con un desarrollo conceptual más ligero, centrado en el intercambio de datos con bajas tasas de envío. La topología es de red en malla, lo que evita la necesidad de tener un computador central que controle al resto de equipos. Al ser más asequible que otros desarrollos similares y tener menos restricciones sobre derechos de comercialización, ha sido el estándar inalámbrico más adoptado por la comunidad *maker*, lo que se ha visto favorecido con la gran cantidad de dispositivos compatibles en el mercado electrónico de bajo presupuesto.

La ligereza del protocolo permite incluso incorporar dispositivos alimentados con batería que pueden durar mucho tiempo sin recarga, facilitando todavía más la penetración de esta tecnología en cualquier hogar.

Ventajas: Es asequible, se basa en una plataforma de código abierto, tiene una gran comunidad que respalda los desarrollos, hay muchos dispositivos e interfaces compatibles...

Desventajas: Es inalámbrico, tiene altas latencias, los dispositivos de fabricantes distintos pueden dar problemas para sincronizarse.

2.5.2. Domótica de integración media

Se trata de sistemas cableados bajo diferentes estándares.

2.5.2.1. X10 (PLC)

Utiliza los propios cables de alimentación de corriente alterna para transmitir la información. Siendo un esquema cableado resulta el más versátil de todos cuando se trata de actualizar domóticamente una vivienda que no fue construida pensando en este tipo de mejora. Aunque se muestra como una opción más, en realidad engloba un gran conjunto de empresas y opciones diferentes (e incompatibles entre sí). El uso de PLC [17] tiene limitaciones importantes. Los

[15] Web comercial de Delta Dore: https://www.deltadore.es/
[16] Enlace de las especificaciones en Wikipedia: https://es.wikipedia.org/wiki/IEEE_802.15.4
[17] Página oficial del protocolo: https://www.x10.com/

equipos conectados deben estar alimentados por la misma fase de corriente, deben situarse *aguas abajo* del mismo interruptor magnetotérmico y la distancia que los separa no puede superar ciertos valores. Además son muy sensibles a todo tipo de interferencias que pueden provocar falsas activaciones y no es recomendable utilizar este canal de transmisión en servicios críticos.

Al estar diseñados para su instalación en viviendas no preparadas para domótica, los transmisores suelen integrarse en el hueco de los enchufes o mandos de la luz encareciendo considerablemente el producto y aumentando la posibilidad de que se produzcan averías.

Ventajas: Versatilidad, mayor fiabilidad que en los sistemas inalámbricos, rapidez de instalación.

Desventajas: Incompatibilidad entre productos de diferentes fabricantes, baja fiabilidad de la comunicación, productos caros o de mala calidad, soluciones cerradas que normalmente se limitan a establecer un enlace alternativo entre un emisor y un receptor.

2.5.2.2. KNX

Es un estándar (ISO/IEC 14543) [18] de comunicaciones en la capa física de OSI, especialmente pensado para domótica. Desde 2016 el estándar está abierto gratuitamente a cualquier fabricante que se suscriba a la asociación KNX. El estándar KNX contempla diferentes medios de transmisión, así que KNX también puede configurar redes de radio (KNX-RF), por alimentación eléctrica (similar al de X10) e incluso en líneas ethernet (EIBnet/IP o KNXnet/IP), aunque la mayoría de las realizaciones, y el factor que diferencia esta familia de dispositivos de las demás es su bus por cableado de par trenzado, mostrado de forma esquemática en el panel de la figura 2.4.

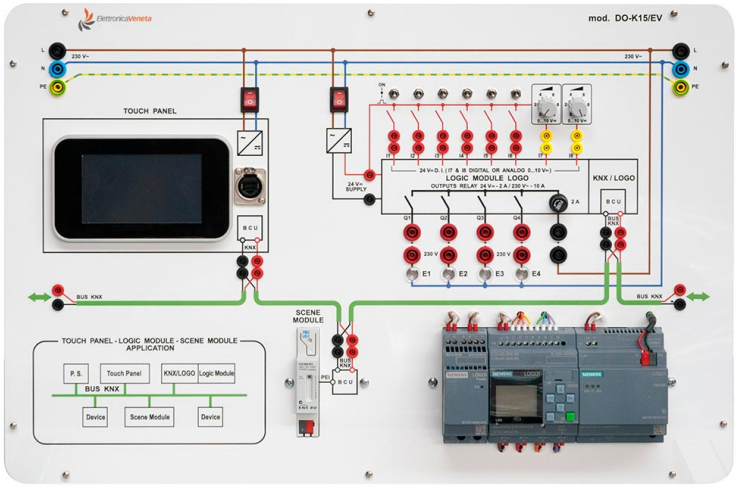


Figura 2.4: Panel de demostración y capacitación para dispositivos KNX

El bus KNX funciona de manera similar a una red de computadores en la que diferentes equipos se comunican entre sí compartiendo el enlace. En el símil, la configuración de red (direcciones

[18] Enlace a la web del consorcio: https://www.knx.org/

IP) se haría desde los propios computadores, pero en KNX es necesario utilizar un software que provee la asociación KNX que sirve para programar todos los dispositivos y establecer los enlaces entre ellos.

Cada dispositivo KNX contiene, por tanto, una memoria programable y perdurable con unos datos de configuración que es necesario escribir para que sea aceptado por la red.

Ventajas: Es una red muy fiable, es muy escalable, tiene detrás un gran número de fabricantes con diversas gamas según presupuesto. La asociación KNX ofrece cursos de certificación que convierte a profesionales técnicos en instaladores homologados, aumentando sensiblemente las opciones de recibir servicio técnico.

Desventajas: Los dispositivos son caros, es necesario crear una red cableada que, en viviendas antiguas obligaría a realizar reformas importantes. La domótica requiere un bus dedicado y exclusivo que no es compatible de momento con ethernet (aunque se está trabajando en una unificación de sistemas), lo que dificulta todavía más la instalación. Los sistemas de control domótico deben seguir las reglas marcadas por la asociación impidiendo los desarrollos de la comunidad de entusiastas. Es imprescindible que un técnico homologado KNX realice la configuración de los dispositivos instalados con las herramientas oficiales, lo que encarece todavía más el producto.

2.5.2.3. Loxone

De prestaciones similares a KNX, Loxone [19] es otro sistema de integración de dispositivos, en este caso en propiedad de una sola compañía. Dispone de una amplia variedad de dispositivos que se conectan mediante un bus propietario especialmente pensado para ello.

Ventajas: Todas las de una red cableada dedicada. Además, al depender de un mismo fabricante, la integración es total y la posibilidad de incompatibilidades es mínima. Loxone también permite formar a todo tipo de profesionales técnicos para convertirlos en integradores, vendedores o reparadores de esta tecnología.

Desventajas: Es un producto costoso, depende de un único fabricante, requiere una red especial dedicada y ofrece un software limitado y propietario.

2.5.3. Domótica de altas prestaciones

Este grupo engloba los estándares y realizaciones que no contemplan restricciones en el presupuesto, exigiendo las máximas capacidades a los edificios en que se integran. En este nivel se desdibuja la línea divisoria entre la electrónica aplicada a las viviendas y la ingeniería civil o industrial existente en inmuebles dedicados a actividades profesionales como fábricas, centrales eléctricas, centros de computación o directamente edificios inteligentes que albergan miles de personas.

[19] Página comercial del fabricante: https://www.loxone.com/

2.5.3.1. Scada

Son las siglas de Supervisory Control And Data Acquisition [20] (Control supervisor y adquisición de datos). Define un software especificamente pensado para controlar y supervisar procesos industriales a distancia. Integrando todo tipo de sensores y actuadores, es capaz de gestionar el proceso automáticamente. Los dispositivos integrados en Scada pueden ser los módulos PLC (programmable logic controller), que realizan la activación de los diferentes aparatos, los PAC (programmable automation controller), que controlan varios PLC estableciendo objetivos de automatización entre ellos, los HMI [21] o puestos de control, para la supervisión y actuación manual sobre el sistema y la red de sensores, preparados para obtener toda la información necesaria en los procesos a controlar. Aunque Scada fue concebido para entornos industriales,

Figura 2.5: Consola centralizada Scada en una vivienda

ciertos edificios inteligentes e instalaciones civiles han confiado la gestión de todos sus elementos activos a sistemas de este tipo, ya que cualquier edificación puede ser contemplada como un entorno industrial, incluso viviendas particulares de alto standing (figura 2.5).

Ventajas: Son sistemas a medida de altas prestaciones, inmunes a problemas de comunicaciones. La arquitectura es abierta y permite todo tipo de automatización, sólo limitada por el presupuesto del cliente.

Desventajas: Es una opción de coste muy alto, dependiente del integrador, los costes de mantenimiento son también muy elevados y al ser a medida, no se pueden exportar las soluciones de unas viviendas a otras.

[20] Enlace a Wikipedia con toda la información sobre el sistema: https://es.wikipedia.org/wiki/SCADA
[21] Interfaz Hombre-Máquina. Es el panel de control del operario para interactuar con un sistema

2.5.3.2. Otros sistemas propietarios

Las viviendas de alto standing han sido objeto de instalaciones que se pueden considerar domóticas, realizadas por técnicos superiores, con el objetivo de mejorar todos los procesos que se pueden automatizar sin reparar en gastos.

Ventajas: La integración con la vivienda es total y la calidad de la solución es máxima. Al ser soluciones a medida es muy difícil manipular los sistemas para burlar la seguridad que proporcionan.

Desventajas: Aparte del coste, la calidad del resultado depende del técnico superior que ha diseñado el sistema y de su acertada elección de tecnología. Existe una importante dependencia técnica con el proveedor que hace difícil actualizaciones futuras o los mantenimientos rutinarios sin contar con él.

2.6. Conclusiones

Casi todas estas opciones tienen alguna de las características deseables en un sistema domótico:

1. **Canal de transmisión**: Que sea fiable, a salvo de interferencias o interrupciones, con gran ancho de banda. KNX, Loxone o Scada serían productos a destacar en este sentido.

2. **Economía**: La domótica no puede ser un lujo o un privilegio, sino una opción. Los dispositivos Zigbee o el ecosistema formado alrededor de los asistentes inteligentes son aconsejables porque permiten una integración gradual, asequible y libre.

3. **Código abierto**: Cualquier tecnología emergente debe implicar a la comunidad de desarrolladores entusiastas. Excluirlos en un producto que pretende llegar a todos los hogares es un error que descarta a la mayoría de fabricantes. Otros, como Zigbee o ITTT permiten tímidamente que los *makers* construyan productos alrededor de estas tecnologías, aunque es un punto muy débil para casi todas las opciones.

4. **Capacidad de integración**: La domótica debe llegar a todos los dispositivos de un hogar para poder coordinar respuestas conjuntas y capturar todas las situaciones posibles ante las que reaccionar. Instalar unas persianas motorizadas y un par de luminarias RGB [22] no es domótica. Sólo los complejos sistemas de las viviendas de alto standing, Scada, Loxone o KNX logran semejantes niveles de integración.

5. **Escalabilidad**: Igual que un ser vivo, un edificio es un ente en evolución. La humanidad es capaz de realizar avances científicos que cada vez son más rápidos y revolucionarios. Los sistemas domóticos no pueden quedar obsoletos ni depender de un único proveedor confiando en que sea capaz de adaptar los nuevos productos al sistema existente. Sólo un estándar abierto en manos de muchos distribuidores o de la comunidad puede asumir la

[22] Dispositivos de iluminación compuestos por diodos led de colores rojo, verde y azul que pueden lucir en cualquier tonalidad cromática variando las proporciones de encendido de sus componentes que forman una base canónica del espectro visible

evolución y adaptarse a ella. Sería estupendo poder hacerlo con Scada o con los productos de KNX, igual que con casi todos los desarrollos de la *domótica ligera*.

La intersección entre todas las necesidades de la domótica y las opciones disponibles es vacía en la fecha de redacción de este texto. A pesar de la increíble cantidad de opciones que hay en el mercado, todavía no ha aparecido un producto o tecnología capaz de iniciar una auténtica revolución capaz de llevar masivamente a los hogares hacia la siguiente etapa, similar a la que experimentó el mundo cuando apareció la luz eléctrica.

Conforme pasa el tiempo surgen nuevas opciones, nuevos productos, nuevos proveedores, nuevos sistemas que, en lugar de ofrecer la solución definitiva, se enrocan en sus objetivos de mercado ampliando todavía más el abanico y despistando a los clientes potenciales que contemplan atónitos cómo el término "*Domótica*" acaba perdiendo su significado en favor de tecnologías que poco tienen que ver con la integración inteligente de un hogar. Todo esto fomenta el desánimo y aumenta la percepción de que un hogar domótico es algo propio de gente adinerada o de *frikis de la tecnología* que viven en un mundo diferente del resto de la humanidad.

La conclusión es que **NO** existe una solución definitiva a día de hoy.

Capítulo 3

Propuesta

El estudio previo evidencia un vacío en la oferta domótica que este estudio pretende llenar con una solución a todos los problemas actuales.

3.1. Requisitos deseables

El objetivo es lograr un punto de comienzo válido e imprescindible para generar aplicaciones domóticas de alto rendimiento que sean asequibles tanto en conocimiento como en presupuesto.

Las prestaciones esenciales son:

3.1.1. Arquitectura abierta

Todos los componentes del sistema deben estar basados en tecnologías accesibles o abiertas. El protocolo y el canal de transmisión tienen que ser fáciles de adquirir, instalar y configurar. El código del controlador principal debe estar realizado en un lenguaje de propósito general ampliamente utilizado que corra contra una máquina virtual y lo haga portable a cualquier sistema operativo. Debe existir documentación a disposición de cualquier usuario para realizar estos programas y una comunidad que respalde todas sus inquietudes, que ponga a su disposición un gran fondo de información en código fuente y documentación.

A nivel hardware debe ser posible incorporar cualquier dispositivo electrónico existente en el mercado permitiendo a cualquier persona desarrollar nuevos drivers para ampliar la cobertura del sistema si fuera necesario sin necesidad de solicitar una licencia ni pagar por ese derecho.

3.1.2. Robustez y fiabilidad

El control de los dispositivos físicos debe realizarse de una forma sencilla, fiable y segura, previendo todos los fallos de comunicaciones, averías de hardware, caídas de las comunicaciones y eventos catastróficos posibles para que incluso en el modo más degradado el sistema domótico siga ofreciendo su servicio a los habitantes de la vivienda inteligente. Estos controladores deben ser fáciles de conseguir y configurar a través de distribuidores independientes y no debe existir ninguna restricción legal ni contractual para que cualquier usuario pueda configurar su propio sistema domótico.

El sistema domótico debe ser capaz de gestionar con total seguridad cualquier instalación civil,

incluso las que comprometan aparatos relacionados con la alimentación eléctrica, calefacción, ventilación, climatización, suministro de agua, gestión de la energía y de las comunicaciones.

El sistema domótico debe superar cualquier intento de anulación maliciosa de su funcionamiento, garantizar una seguridad antirrobo, prevenir los incendios, las inundaciones y los posibles escapes de gas. Debe disponer de mecanismos alternativos en los procesos críticos que permitan mantener bajo control y con total seguridad la situación incluso en caso de fallos de comunicaciones, de energía y otras situaciones catastróficas hasta el límite de las posibilidades de los dispositivos físicos involucrados.

3.1.3. Escalabilidad

Dado que la tecnología está en permanente evolución, el sistema debe garantizar el funcionamiento correcto de la instalación actual con el paso del tiempo manteniendo compatibilidad hacia atrás sin excepciones y permitir integraciones mixtas con controladores, unidades principales o dispositivos de tecnologías más avanzadas sin que se resientan las prestaciones de todo lo instalado previamente.

Esta compatibilidad debe extenderse hacia cualquier fabricante o distribuidor adherido al estándar, que garantizará que sus productos cumplirán todas las especificaciones de diseño liberando indefinidamente al cliente de ataduras con una marca o producto concreto.

3.1.4. Accesibilidad

El sistema domótico debe ser transparente a cualquier tipo de usuario, incluso en las realizaciones más complejas, ofreciendo un entorno amigable en el que sea la propia automatización la que se encargue de gestionar las interacciones entre dispositivos. En caso de errores o averías, el propio sistema domótico debe ser capaz de adaptar sus funcionalidades inhibiendo los componentes defectuosos si es necesario, indicando al usuario la existencia del problema, pero librándole de la carga técnica de tener que corregirlo. En su lugar, establecerá un canal alternativo de gestión del mantenimiento o servicio técnico con personal cualificado dando a escoger al cliente con qué proveedor de soporte quiere resolver el problema.

El hogar domótico debe ser capaz de adaptarse a las limitaciones físicas de todos los habitantes, especialmente de los niños, ancianos y aquellos que tengan limitada alguna de sus facultades, anticipando y previniendo cualquier situación de riesgo potencial.

3.1.5. Eficiencia

Es imprescindible que los dispositivos del hogar cooperen de forma conjunta para alcanzar el máximo grado de confort y seguridad para los habitantes, de la forma más eficiente y sostenible. Para ello el sistema responderá ante peticiones de objetivos y no de especificaciones. Esto quiere decir que si un usuario desea luz en el salón para leer, al hacérselo saber a la domótica, ésta deberá procurar el objetivo deseado utilizando el medio más eficiente: Si es de día y no hace

mucho calor, el sistema abrirá la persiana del salón y si no, encenderá una luz.

Cualquier implantación domótica deberá relacionar los costes de consumo de energía con su propio funcionamiento aprovechando las tarifas más ventajosas o gestionando los medios de autoabastecimiento disponibles (si los hay), de forma transparente al usuario que sólo debe notar la presencia domótica en el ahorro de su factura mensual.

3.1.6. Separación por capas

Igual que en todas las disciplinas que relacionan dos o más áreas científicas como la **robótica**, que integra un desarrollo industrial, una implementación de software y a veces una capa de inteligencia artificial, es imprescindible demarcar claramente las fronteras entre ellas. La domótica involucra al integrador de hardware, cuya misión es proporcionar un equipo fiable y versátil tolerante a fallos, al técnico que desarrolla los sensores o actuadores, al profesional que realiza el proyecto con un hogar específico y al desarrollador de software que concibe el modo de interactuar del producto final. Cada una de estas funciones es independiente de las demás y requiere un perfil muy específico.

El sistema domótico debe tener en cuenta todos estos puntos de vista y proporcionar un entorno de desarrollo en el que cada participante pueda aportar en su área específica, de la misma forma que un proyecto informático se divide en tareas independientes. Es misión de la domótica definir claramente estas funciones y establecer las fronteras de una forma precisa.

3.1.7. Disciplina/Convenio

El peor problema de la falta de coherencia de las opciones actuales en domótica se llama dispersión. Es el efecto de la *Torre de Babel*, que cada una de ellas trata de resolver los mismos problemas de formas diferentes. Es imprescindible sentar las bases del estilo de programación, de diseño y hasta de presentación para que la comunidad pueda construir usando como base desarrollos previos, trabajando así en la misma dirección. Compartir una arquitectura hardware es sólo el cimiento de una pila de componentes compatibles, fáciles de mantener, bien documentados, con licencias abiertas y capaces de evolucionar en manos de otros desarrolladores diferentes de los que concibieron el modelo original.

3.2. Características del sistema propuesto

Definir las bases de una estructura documental que debería ser concebida y consensuada entre un gran número de personas expertas en sus respectivos campos de trabajo se sale del ámbito de este proyecto. No obstante se va a proponer una arquitectura básica que va a cobrar sentido a través de una metáfora de la mitología romana clásica.

En la cultura latina, el hogar desde el punto de vista del desarrollo de la existencia humana se consideraba algo mucho más complejo que un simple refugio a resguardo del mundo exterior.

La vida en sí se desarrollaba en el escenario doméstico y la casa era un protagonista más que tenía sus propias deidades particulares, mostradas en la figura 3.1 que eran los *manes*, que eran las almas de los familiares difuntos, los *lares*, que protegían sus respectivas casas y los *penates*, encargados de los alimentos y la salud. La domótica encaja perfectamente en la función de los lares; por eso en la solución propuesta, el código de computador que define el comportamiento de un hogar inteligente recibe este nombre.

Figura 3.1: Deidades romanas del hogar

En la mitología, la diosa Hestia o **Vesta** (figura 3.2), hija de Cronos y Rea, regía sobre el fuego del hogar, la cocina y la arquitectura. De alguna forma su obra dotaba a la vivienda de una presencia benigna y protectora. Se ha pensado que para ejercer un control solvente entre la gestión a nivel lógico del código de los **lares** y los aparatos eléctricos o electrónicos que conforman las partes activas de un edificio, es necesario establecer un puente que los relacione y que sea capaz de gestionar de forma correcta cualquier contingencia que pudiera afectar a estos últimos sin perder la funcionalidad del primero.

Figura 3.2: La diosa Vesta

3.2.1. Controlador Vesta

Debe ser un circuito basado en un microcontrolador de bajo coste, suficientemente sencillo como para no precisar un sistema operativo y así eliminar cualquier latencia de arranque y de respuesta. Tiene que ser capaz de gestionar correctamente cualquiera de los posibles dispositivos físicos existentes y futuros, asumiendo sus servidumbres y filtrando sus defectos al comunicar con **Lares**.

Para evitar el encarecimiento y una complicación excesiva en la instalación doméstica, cada controlador Vesta debe poder controlar varios dispositivos físicos simultáneamente, estableciendo *hubs* de enlace aprovechando cajas de empalmes o armarios eléctricos existentes en el edificio.

Debe ser capaz de mantener en funcionamiento todos los dispositivos a su cargo en situaciones degradadas o incluso catastróficas.

El canal y el protocolo de comunicación con la central domótica debe ser estándar. Se utilizará una red cableada ethernet como la existente en numerosas viviendas, que además se puede compartir con los computadores de la vivienda y los teléfonos Vo-IP[1]

Los posibles dispositivos a controlar se clasificarán por familias homogéneas atendiendo a la afinidad de funcionamiento. Por ejemplo, la familia *Salida Binaria* concebida como un dispositivo que puede tener dos estados (conectado / desconectado) se aplicará a las lámparas, a los ventiladores, a bombas de fluidos, sirenas, etc. Todos ellos comparten el funcionamiento aunque sean de naturaleza muy diferente. Las diferencias elementales entre los dispositivos que pertenecen a una misma familia se resolverán mediante parámetros de configuración. En el caso de una *Entrada Binaria*, que engloba cualquier dispositivo que informa de su estado al sistema, se pueden tener circuitos con una salida lógica estable, como los sensores crepusculares o de presencia y también interruptores o pulsadores mecánicos que acusan el defecto de los rebotes[2] y para resolverlo se suelen ajustar unos temporizadores que desactivan la detección durante unos milisegundos asumiendo que al final de ese tiempo ya habrán terminado todos los rebotes y la señal estará estabilizada. Como no todas las entradas tienen este problema, se introducirá un parámetro de configuración en cada una de ellas donde se activará o no el temporizador según sea necesario.

Cada dispositivo tendrá asignada una dirección única y constante dentro del controlador y se deberá direccionar externamente por medio de un mnemónico que asignará el usuario editando la configuración. Desde el controlador domótico **Lares** se podrá direccionar cualquier dispositivo utilizando este valor mnemónico. Se resolverán todos los posibles valores duplicados de forma que siempre se pueda seleccionar un dispositivo concreto.

En caso de fallo en las comunicaciones o caída del sistema el controlador debe contar con un modo degradado o modo desconectado, en el que los eventos de los dispositivos se procesarán de forma local según lo que indique el usuario en los parámetros de configuración. En este modo

[1] Conjunto de recursos que hacen posible que la señal de voz viaje a través de internet empleando el protocolo IP. La señal de voz se envía en forma digital, en paquetes de datos.

[2] El rebote eléctrico o traqueteo es un problema común con interruptores y relés. Cuando los contactos se tocan su impulso y elasticidad actúan de forma que éstos rebotan una o más veces antes de que se dé una unión fija. El resultado es una rápida corriente eléctrica en vez de una transición clara de cero a una corriente que fluye totalmente.

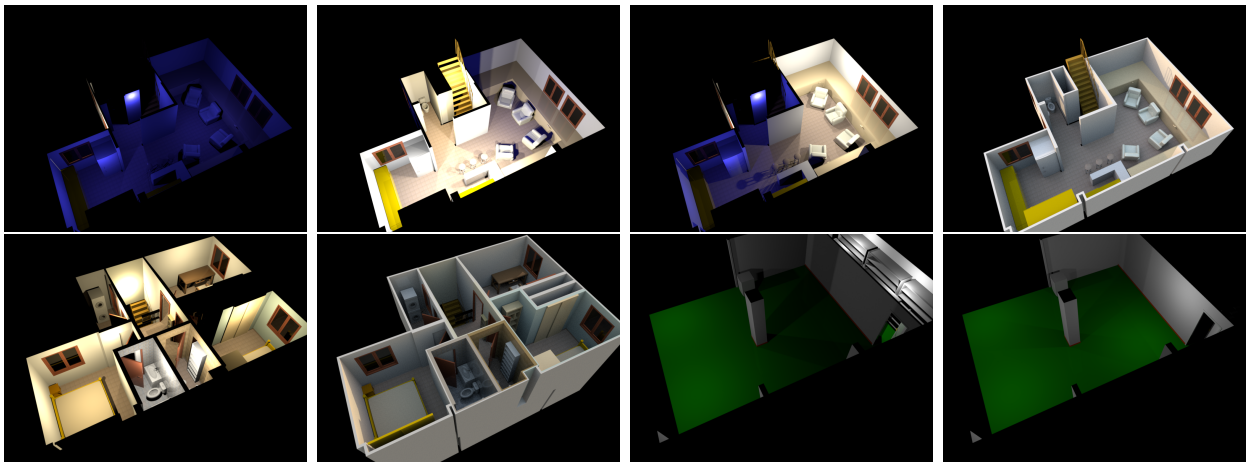


Figura 3.3: Diversas infografías creadas para una versión previa de la domótica de este hogar

local un dispositivo podrá responder a los eventos generados por otro perteneciente a la misma controladora Vesta introduciendo como parámetro su dirección única.

Aparte de los parámetros de configuración propios del tipo de dispositivo, todos ellos tendrán en común los siguientes:

1. Mnemónico: Nombre o identificación de cada dispositivo. Por ejemplo: "LuzCocina"
2. Enabled: Cada dispositivo se puede habilitar o deshabilitar. Los dispositivos de entrada dejarán de desencadenar eventos y los de salida cesarán en su función cuando estén deshabilitados.
3. Service: El modo de servicio deshabilitará la gestión automatizada de los dispositivos y permitirá modificar cualquier parámetro, incluso los que se extraen del dispositivo físico. Servirá para hacer pruebas, realizar mantenimientos y emitir diagnósticos de un dispositivo concreto.
4. Internal Events: El dispositivo desencadenará eventos cuando ocurran sucesos en el hardware, que se gestionarán de forma interna en la propia controladora.
5. External Events: Lo mismo, pero para generar paquetes UDP que informarán al sistema central y al resto de las controladoras.
6. Manage Events: Habilitará o deshabilitará la recepción de eventos desde otros dispositivos en el modo local.
7. Auto Local: Pasado un tiempo sin comunicación desde el procesador Lares, el dispositivo pasará a funcionar de forma local (comportamiento en respuesta a eventos internos).
8. Auto Remote: Cuando la controladora reciba un paquete de datos desde Lares, el dispositivo recuperará su función normal (dejará de escuchar los eventos internos y sólo atenderá órdenes de Lares).

La propia controladora Vesta se configurará como un dispositivo y siempre tendrá la dirección cero. Los parámetros comunes de configuración enumerados afectarán a todos los dispositivos contenidos en ella.

El mnemónico de la controladora se utilizará como prefijo a la hora de direccionar cualquier dispositivo desde Lares. Si se tuviera una controladora llamada *ArmarioElectrico* y se pretende encender la luz de la cocina desde Lares habrá que emplear la sentencia:

```
ArmarioElectrico.LuzCocina.Switch(true);
```

La controladora Vesta debe almacenar y gestionar los datos sobre los dispositivos que contiene y sus respectivos valores de configuración. El usuario tiene que poder acceder y escribir estos valores de forma sencilla mediante una herramienta de terminal conectada al puerto serie del microcontrolador o a través de HTML desde cualquier otra dirección de la red ethernet.

La tabla de dispositivos y sus valores de configuración deben ser accesibles desde cierto software que con él creará la estructura personalizada y de forma automática de la instalación Lares de cada vivienda.

3.2.2. Procesador domótico Lares

Será un programa escrito en un lenguaje orientado a objetos de propósito general que debe correr contra una máquina virtual que tenga acceso a la red Ethernet. Se compondrá de una parte común a todas las instalaciones domóticas que se encargará de las rutinas de transferencia de información, gestión de conexiones, almacén de referencias de las controladoras Vesta y agenda de sucesos, y otra específica para cada hogar, que será generada de forma automática por un compilador cuyo código fuente será la lista de las direcciones en la red de las controladoras Vesta instaladas.

El compilador generará una clase abstracta que contendrá una colección de objetos, que serán las imágenes lógicas de los dispositivos físicos instalados en todas las controladoras. Esta clase representará el hogar.

La clase abstracta de Lares tendrá acceso a:

1. Los dispositivos instalados en los controladores Vesta (recordar que la propia tarjeta controladora se considera como un dispositivo más).
2. Dispositivos Vesta virtuales: Son componentes gestionados por el sistema que por su naturaleza no se pueden conectar a una controladora física, pero tienen la misma funcionalidad.
3. Una agenda de sucesos en la que se pueden guardar llamadas a función que se producirán en un momento específico.
4. Un almacén persistente de datos.
5. Un módulo de comunicaciones a través del cual se pueden hacer consultas en los protocolos de comunicación habituales (servicios RSS, peticiones SOAP, consultas en WEB, ...)

Complementando a Lares puede existir una herramienta de visualización HMI similar a las de los sistemas Scada, que extraiga de la base de datos gestionada por Lares el estado de los

dispositivos y pueda visualizar su situación mediante esquemas gráficos como los de la figura 3.3.

3.2.3. Planes domóticos

Se llamará *Plan domótico* a cualquier objeto que herede de la clase abstracta de Lares. Cada plan domótico tendrá uno o más delegados en respuesta a los eventos de los dispositivos que a su vez desencadenarán acciones. Los planes domóticos sencillos reaccionarán a los eventos siempre de la misma forma emulando el funcionamiento de una vivienda clásica.

Por ejemplo, se puede generar una función en respuesta al evento

```
ArmarioElectrico.LuzCocina.onChange(bool value)
{
    ArmarioElectrico.PlafonTecho.Switch(value);
}
```

De esta forma, el interruptor quedará asociado al plafón del techo de la forma tradicional.

Al definir el comportamiento del sistema por medio de código en un lenguaje de programación, las posibilidades se multiplican con todas las estructuras de control, la posibilidad de almacenar valores previos, la relación entre varios parámetros, el uso de librerías de funciones con comportamientos domóticos, etc.

3.3. Conclusión

El sistema propuesto establece un puente entre el mundo físico y el lógico, pudiendo abordar cada uno de ellos de una forma concreta. Así, los técnicos, *makers* y profesionales instaladores podrán centrarse exclusivamente en la colocación, configuración en Vesta y pruebas de los aparatos a domotizar.

Al mismo tiempo, los programadores realizarán desarrollos de software capaces de modificar y mejorar el sistema de la misma forma que se desarrolla, mantiene y depura cualquier otro programa. Dada la separación existente entre la parte física y la lógica, es posible crear una simulación de la vivienda para probar las nuevas versiones del código mucho antes de desplegarla en la vivienda real, estableciendo una nueva forma de concebir la domótica como ciencia, más centrada en el objetivo que en el marketing y preparada para realizar sobre esta base cualquier investigación.

Tanto el programa Lares como los dispositivos Vesta se encargarán de hacer que este puente se comporte como se espera, gestionando de forma efectiva todos los problemas que pueden ocurrir con el hardware, para que los expertos en hardware y los expertos en software se encarguen de sus respectivos trabajos.

Capítulo 4

Diseño

Siendo éste un proyecto que involucra componentes hardware y una variedad de tecnologías, los numerosos riesgos impiden una planificación estática secuencial. Al tratarse de un proyecto desarrollado por un solo programador, el único método posible es el planteamiento de objetivos parciales materializados en prototipos, lo que sugiere un modelo de planificación en **espiral**. Además de los propios riesgos técnicos, típicos de la combinación de diferentes tecnologías, son especialmente preocupantes los relacionados con el rendimiento y el comportamiento en campo de los dispositivos de control, lo que obliga a añadir pruebas de laboratorio a las propias de un desarrollo puro de software.

4.1. Alcance del proyecto

Dado lo inabarcable que es crear toda una arquitectura domótica y que se ha asumido que el trabajo pretende establecer las bases en torno a las que se construirá el resto del ecosistema, se establecen los objetivos en la realización de una controladora hardware llamada **Vesta** y un programa de control de controladoras denominado **Lares**.

En la primera interacción se desarrollará un prototipo funcional de módulo Vesta.

4.2. Análisis de requerimientos

Las condiciones a cumplir son:

1. El soporte hardware debe ser un microcontrolador de bajo coste.

2. El módulo Vesta debe gestionar una cantidad indeterminada de dispositivos físicos de forma concurrente y controlada.

3. Se asume que el canal de comunicaciones que une el controlador Lares con cada controlador Vesta puede ser compartido con otros sistemas de forma que hay que prever reacciones ante posibles sobrecargas del ancho de banda disponible y hacer un uso responsable de este recurso limitado. **No** se permitirá el pooling[1] siendo preferible una gestión efectiva de

[1]Estrategia de muestreo en la que un sistema comprueba repetidamente el estado de un dispositivo haciendo llamadas a intervalos regulares de tiempo. Se considera que el pooling supone un uso innecesario del canal de transmisión.

eventos, el uso de interrupciones y la generación de pulsos de control o latidos mediante técnicas de watchDog[2].

4. Deben establecerse dos modos de funcionamiento: Local y remoto, conmutando entre ellos en función del estado del canal de enlace.

5. Cada módulo Vesta debe disponer de una forma sencilla y efectiva para consultar el estado de funcionamiento, modificar los valores de estado y editar los parámetros de configuración.

Se va a seguir la metodología de modelado incremental con desarrollo de prototipos.

4.3. Casos de uso

La interacción de un actor humano con el sistema sólo se puede realizar de tres maneras que definen otros tantos casos de uso.

1. Interacción domótica: Se da durante el uso normal, cuando un habitante de la casa interactúa con su vivienda a través de sus acciones.

2. Consulta mediante web: Se prevé una interfaz genérica mediante HTML que permitirá al usuario o al servicio técnico el acceso remoto a la domótica para formular consultas, peticiones o alterar la configuración.

3. Configuración de un módulo Vesta: El actor es el técnico de despliegue del sistema, con el objetivo de adaptar o calibrar la controladora con los dispositivos físicos que ha de gestionar.

4.3.1. Interacción domótica básica

Mostrado en la tabla 4.1, es imposible prever todos los desarrollos domóticos que pueden ser tan simples como una simple reacción a un evento o tan complejos como un algoritmo de aprendizaje automático que usa la interacción del usuario como refuerzo para alimentar el almacén de datos. Lo interesante de este caso de uso es que la existencia del sistema es transparente al usuario típico. Su actuación en el domicilio no se concibe en ningún caso como una interacción con un sistema informático sino como una parte cotidiana de la experiencia de habitar una casa.

4.3.2. Consulta mediante Web

Los navegadores Web están presentes ya en todas partes, desde los propios computadores hasta los teléfonos móviles. Permiten una comunicación cómoda y uniforme que es independiente del

[2]Un temporizador de perro guardian es un mecanismo que previene y detecta un fallo en las transmisiones o en los equipos. Al terminar la cuenta, el temporizador reiniciará el equipo bajo control o pasará a un modo degradado. Para evitarlo, el equipo protegido enviará señales periódicamente que reiniciarán la cuenta impidiendo que llegue a su final.

Usuario	Sistema
Conecta un interruptor	
	Enciende la luz
Desconecta el interruptor	
	Se apaga la luz

Tabla 4.1: Caso de uso para una interacción domótica básica

sistema operativo o el canal de transmisión, de forma que se pueden usar tanto en el propio domicilio como a distancia. Dadas las peculiaridades del servicio Web, es necesario filtrar el acceso a la información y los recursos del hogar por medio de algún sistema aún por determinar, aunque no es relevante en esta etapa. En este caso de uso las credenciales serán el identificador del usuario y una contraseña Este caso queda descrito en la tabla 4.2.

Usuario	Sistema
Teclea la dirección del servicio en el navegador Web	
	Carga de la página de inicio de la casa
Pulsa el botón *Login*	
	Muestra un cuadro de diálogo con el id. de usuario y la contraseña
Introduce los datos	
	Muestra la página principal de la casa
Modifica un valor (opcional). Este paso se puede realizar varias veces.	
Pulsa el botón de enviar.	
	El sistema actualiza los datos cambiados y los muestra

Tabla 4.2: Caso de uso para el acceso remoto a Lares

4.3.3. Configuración de Vesta

Se realiza tras conectar el módulo con los dispositivos bajo control y cada vez que es necesario realizar ajustes. El actor siempre será el personal de servicio técnico o el instalador. Sólo será el usuario si además ejerce alguna de las otras dos funciones (Tabla 4.3).

Usuario	Sistema
Teclea la dirección de la controladora en el navegador HTML	
	Carga de la página de inicio de la controladora
Selecciona el parámetro elegido	
Modifica el valor (opcional). Este paso se puede realizar varias veces.	
Pulsa el botón de enviar.	
	El sistema actualiza los datos cambiados y los muestra

Tabla 4.3: Caso de uso para la configuración de Vesta

4.4. Estructuras de datos

En la primera interacción se hace necesario definir la estructura básica de gestión de los dispositivos bajo control en las controladoras **Vesta**. Es importante considerar los condicionantes de espacio y de eficiencia de un microcontrolador básico a la hora de dimensionar esta estructura.

4.4.1. Familia de tipos

Cada dispositivo Vesta engloba un conjunto de aparatos físicos con un modo de uso equivalente en el que se definen ciertos parámetros que permiten adaptarse a las condiciones particulares de cada uno de ellos. Ya que apenas existen tipos o familias de aparatos atendiendo a su funcionalidad y debido a las restricciones operativas citadas, se ha decidido que el valor que decidirá el tipo estará definido en un byte. Se reserva el tipo 0 para el dispositivo *Controladora*, los tipos 1 a 127 serán destinados a dispositivos de propósito general (entradas binarias, salidas binarias, entradas numéricas, salidas numéricas, puertas lógicas, termostatos,...).

Los tipos 128 a 253 se reservan para dispositivos de uso específico para aplicaciones reservadas a clientes industriales, asumiendo que los valores asignados sólo definen estos tipos a nivel de

despliegue, pudiendo ser diferentes en diferentes distribuidores.

El tipo 254 es un valor reservado para las plantillas de código y Vesta los ignorará.

El tipo 255 se define como *Tipo Nulo* y Vesta lo ignorará.

4.4.2. Tabla de slots

Cada controlador Vesta contendrá una tabla que permitirá gestionar todos los dispositivos que ésta contiene. Dado que es posible que haya peticiones simultáneas, es necesario definir la prioridad del funcionamiento de los dispositivos en una ejecución concurrente. A esta prioridad la denominaremos slot.

La tabla de slots contendrá el número de slot asignado, el tipo de dispositivo y un punto de acceso a los datos específicos de la instancia de todos los dispositivos conectados a ese controlador Vesta. El número de slot cero quedará asignada al propio controlador. El número 255 se considerará un dispositivo nulo.

Esto quiere decir que si existen dos peticiones simultáneas para atender, el sistema procesará primero la del dispositivo con el número de slot más bajo. De esta forma, al tener asignado el slot 0, la controladora siempre tendrá la máxima prioridad. El sistema debe utilizar la combinación del número de serie de la controladora Vesta y el número de slot para identificar de forma única a cada dispositivo del sistema. Sin embargo, desde la vista de código (en Lares), se utilizará el identificador que el usuario puede editar traduciendo los nombres por los códigos en una tabla de resolución.

4.4.3. Memoria de configuración de dispositivos

Cada dispositivo se compone de una parte común al tipo o familia de dispositivos y otra específica de cada instancia. Esta parte específica de la instancia consiste en tres estructuras de almacenamiento que se muestran en la tabla 4.1 y se describen a continuación.

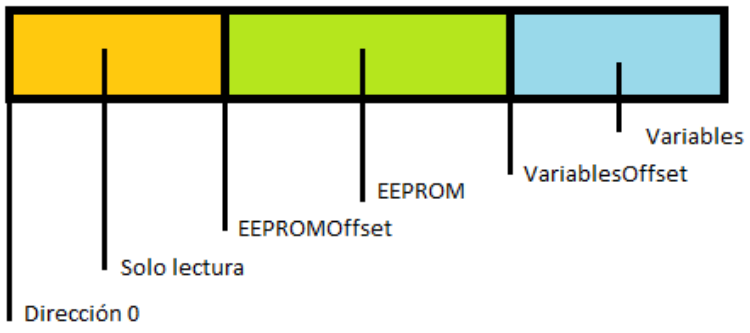


Figura 4.1: Las tres zonas de almacenamiento

4.4.3.1. ROM

Contiene información de la instancia del dispositivo que se asigna en fábrica y no cambia durante toda la vida útil de la controladora Vesta. En el caso de dispositivos asociados a pines del microcontrolador, el número de pin, que en cada instancia está conectado siempre al mismo, es un valor que residirá en este tipo de memoria.

Se implementará mediante una constante estática de tipo array.

4.4.3.2. EEPROM

Esta memoria funciona como la ROM en el sentido de que los datos contenidos se conservan incluso tras cesar la alimentación, pero se admite que el usuario realice cambios ocasionales[3]. El identificador del dispositivo, los flags de estado y ciertos parámetros de configuración residen en este tipo de memoria.

4.4.3.3. RAM

Es donde se almacenan los datos que pueden cambiar y cuyo valor se pierde al reanudar el funcionamiento del microcontrolador. Se usan para guardar los valores actuales asociados al dispositivo y las variables de estado necesarias para su funcionamiento. En las salidas temporizadas el valor de tiempo restante se almacenará en RAM, igual que el estado actual de activación.

Cada entrada de la tabla de slots contiene además tres campos con los valores de las direcciones de inicio en cada uno de estos tres espacios de memoria. Dado que son memorias de tipos distintos, su numeración no es correlativa. El slot 0, que es el propio controlador comienza en la dirección 0 de RAM, la dirección 0 de ROM y la dirección 0 de EEPROM. El segundo dispositivo comenzará en las direcciones resultantes de sumar el espacio en ese tipo de memoria más uno y así consecutivamente.

4.4.4. Acceso a los datos

La tabla de slots y los respectivos espacios de memoria permiten accesos de distinto nivel a la información de la controladora. Las rutinas internas escribirán en las direcciones pertinentes por medio de los algoritmos del dispositivo al que pertenezcan, la interface web de configuración accederá a ellos sin necesidad de invocar el código del dispositivo con sólo saber la estructura, el tipo y el orden de colocación de cada elemento de datos. Esto permite desplazar toda la gestión de datos al FrontEnd[4] agilizando en gran medida la respuesta, pues los equipos del lado del cliente siempre serán más potentes que el limitado hardware del servidor.

[3] Ver casos de uso/ configuración del módulo Vesta

[4] En las arquitecturas cliente - servidor con tecnología Web, se denomina FrontEnd al proceso que realiza el computador del cliente y que se ejecuta en un lenguaje de scripting como JavaScript contra una máquina virtual que proporciona el propio navegador.

Capítulo 5

Desarrollo del proyecto

El objetivo de este proyecto es obtener un controlador domótico basado en una arquitectura asequible y de código libre. Por eso se escogió Arduino. En principio se contó con el modelo **Arduino Uno** por ser el más distribuido y el que se utiliza en todos los equipos de iniciación. Existe abundante documentación y una nutrida comunidad de desarrolladores, además de que casi todos los **shields**[1] están concebidos para esta placa. Sin embargo el imperativo de contar con una interfaz web disparaba el tamaño del programa muy por encima de los 32Kb de memoria disponible. El resto de los valores y el reducido número de pines de control para dispositivos terminó de desaconsejar el montaje. Una de las claves de Vesta es centralizar un cierto número de aparatos en cada controlador para emplear la cifra más baja posible de ellos.

Por eso se eligió el modelo **Arduino Mega** que con sus 256Kb de memoria flash, 4Kb de EEPROM y 8Kb de RAM estática encajaba sobradamente con las capacidades requeridas. Además dispone de 16 pines de entradas analógicas, 16 de salidas pwm[2] y hasta 54 pines que pueden ser configurados como entradas o salidas digitales. Además es compatible con todos los shields concebidos para el Arduino Uno.

Unida con la tarjeta principal se ha de agregar un shield de ethernet basado en el chip w5100[3] Aunque muchos de estos shields están preparados para albergar una memoria SD se ha optado por no hacer uso de esta posibilidad para mantener la lista de requisitos tan baja como fuese posible y eliminar posibles fuentes de avería dado el entorno de trabajo en que deberán funcionar estas controladoras.

5.1. Desarrollo de las placas Vesta

En esta sección comentaremos las diferentes iteraciones realizadas durante el desarrollo de las placas Vesta.

5.1.1. Primera iteración

Se utilizó el IDE Visual Studio Code con un plugin que lo enlaza con el compilador de Arduino, inferior en prestaciones al primero. Se comenzó generando las estructuras de datos previstas

[1]Placas de expansión cuyos terminales están colocados en la misma disposición que en la placa principal de forma que se puedan apilar.

[2]Pulse Width Modulation o modulación por ancho de pulsos es un tipo de salida digital que combinada con un sencillo filtro de condensador puede proporcionar una salida de tensión analógica

[3]Wiznet W5100 es un procesador elemental capaz de establecer cuatro puertos ethernet de forma simultanea. https://www.sparkfun.com/datasheets/DevTools/Arduino/W5100_Datasheet_v1_1_6.pdf

en la fase de diseño. Para separar la parte de datos de la de gestión de cada dispositivo, se crearon dos clases por cada uno de ellos llamadas *xxxConfig* y *xxxItem* siendo *xxx* el nombre de la familia de dispositivos. Pronto aumentó la cantidad de clases y de archivos del proyecto ganando en tamaño a medida que se iban teniendo en cuenta las peculiaridades de cada uno.

Con un pequeño número de dispositivos desarrollados y una versión funcional del primer prototipo del núcleo se comenzó a desarrollar el servidor web para la configuración. Para mejorar la presentación se añadió a la plantilla de la página una referencia al framework Bootstrap[4]. Las páginas se generaban de forma estática (figura 5.1). Cada tipo de objeto componía su presentación y la respuesta a las peticiones encadenando llamadas a una instancia de un objeto que devolvía los códigos HTML de los diferentes objetos gráficos del framework.

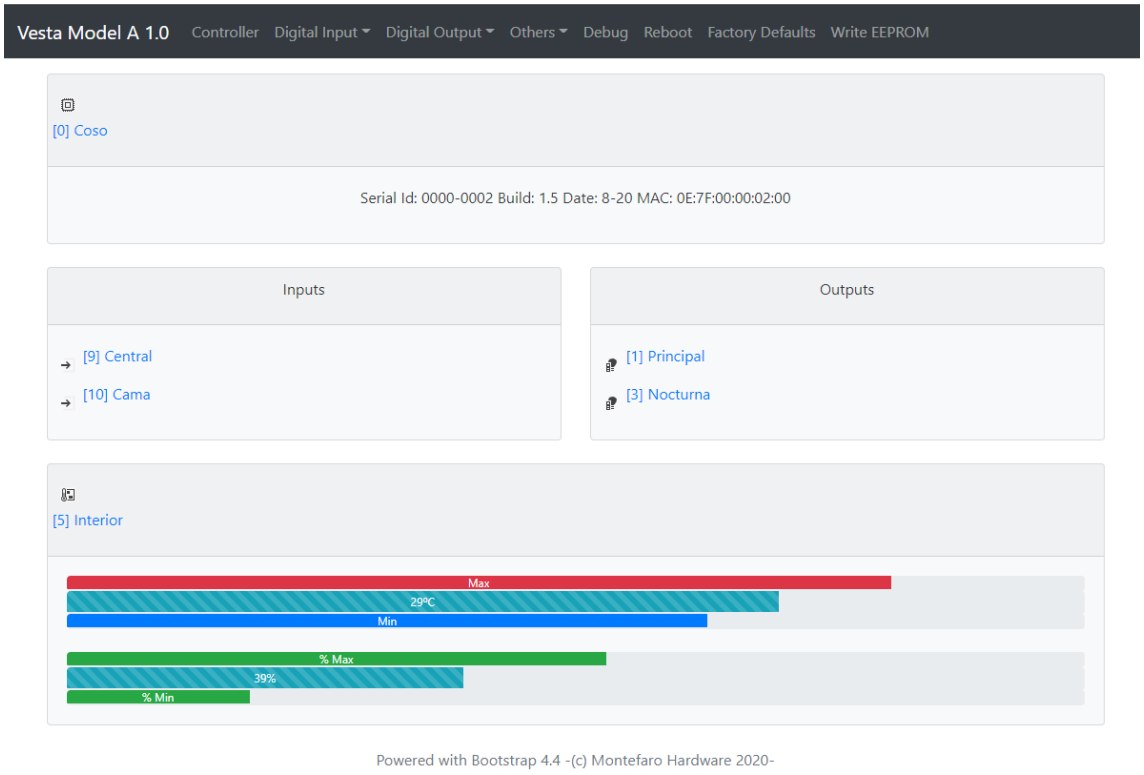


Figura 5.1: Interface Web con Bootstrap de Vesta en 2020

Pruebas

Los tiempos de respuesta del servidor web eran excesivos. Generar la respuesta a las peticiones HTML mantenía ocupado el procesador de Arduino que dejaba de gestionar los dispositivos mientras se encargaba de ello. El problema de la latencia se agravaba porque a menudo excedía el timeout del navegador y a menudo bloqueaba el puerto correspondiente del chip W5100 que quedaba inservible hasta que se reiniciaba la placa.

Se intentó rebajar la carga de trabajo simplificando la interfaz, lo que hizo posible que la placa

[4]Es una librería de código JScript y hojas de estilo CSS que mejora notablemente los objetos HTML nativos añadiendo funciones de presentación y edición desde el lado del cliente. https://www.w3schools.com/whatis/whatis_bootstrap.asp

se comportase de forma aceptable en dispositivos con pocos parámetros de configuración, pero en las vistas generales que combinaban una presentación conjunta de los dispositivos instalados se volvía a producir el problema del timeout del navegador y otras veces el bloqueo del puerto del W5100.

Para no prolongar esta fase del desarrollo se liberó una versión de pruebas con un número reducido de dispositivos (cuatro entradas binarias, cuatro salidas binarias y un sensor de temperatura y humedad). Se tuvo esta placa en funcionamiento durante dos meses integrada en la domótica existente en la casa sin más fallos reseñables que los del servidor Web.

Era evidente que había que mejorar la parte del servidor Web y corregir el problema de la inutilización del puerto ethernet que acababa dejando fuera de linea la tarjeta tras agotar los disponibles tras varios fallos en el servicio HTML. Es importante reseñar que estos fallos se podían producir también por causas ajenas al propio servidor como la interrupción de la carga desde el propio navegador, que es una circunstancia probable contra la que debería existir una respuesta.

Otro fallo que se detectó posteriormente era la entrada en bucles infinitos cuando la configuración de los dispositivos provocaba que el sistema quedara esperando indefinidamente ante el cambio de estado de una entrada binaria o cuando se producía un evento cíclico que desencadenaba un evento. En estos casos debería producirse un reinicio de la controladora.

Además de los errores reseñados hubo que corregir y tratar con otra gran cantidad de problemas debido a errores lógicos de programación, algunos de ellos muy difíciles de detectar porque la placa Arduino Mega no cuenta con ninguna herramienta de depuración en línea.

5.1.2. Segunda iteración

Para ayudar en la depuración de los errores futuros se desarrolló un módulo de respuesta a entradas de texto por el puerto serie creando un terminal en el que se podían ejecutar diversos comandos, como listas de dispositivos, consultar el estado actual, las posiciones de la memoria y sacar un pequeño histórico de las modificaciones en ciertas variables compartidas.

Además se diseñó una rutina de control de bloqueo de puertos Ethernet utilizando código de una web[5]. La función se invocaba tras un tiempo ajustable como parámetro del controlador, revisaba el estado de los puertos y llamaba a las funciones de la librería de gestión del w5100 reiniciando los que encontraba en un estado abierto cuya marca temporal había transcurrido más allá de un plazo aceptable desde el momento actual.

Se simplificó la librería del servidor Web sustituyendo la composición del código HTML mediante encadenamiento de funciones por constantes literales tras comprobar que la invocación de funciones desde otras funciones en Arduino consume mucho más tiempo de procesador que el envío de constantes de texto ya hecha. En este punto cada compilación arrojaba mensajes de aviso informando de la escasez de memoria estática para el código resultante que podían ocasionar corrupción de datos.

Se habilitó el perro guardián (watchdog) por hardware que lleva el procesador Atmel de Arduino

[5] https://forum.arduino.cc/t/ethernet-shield-w5100-stops-responding/442776

para evitar el bloqueo de la placa al entrar en bucles infinitos y se añadió el correspondiente flag de activación o desactivación y se incorporó un nuevo mecanismo de gestión de interrupciones por hardware que el Arduino Mega tiene disponibles en algunas de sus entradas digitales. De esta forma éstas podrían considerarse como dispositivos de alta prioridad fuera del bucle de funcionamiento lo que mejoraría la eficiencia al obviar estos dispositivos del pooling y permitiría responder muy rápidamente ante estos eventos incluso aunque el servidor HTML ocupase todo el proceso. En realidad lo que se buscaba era una alternativa al problema del servidor HTML para mantener el servicio incluso en el peor de los casos. Igual que en la primera interacción no era asumible tener parado el proyecto sólo por un fallo, por graves que fuesen sus consecuencias.

Pruebas

El mayor problema vino de la mano del cálculo manual de las direcciones en que se almacenaban los parámetros de cada tipo de dispositivo, incluso con la ayuda de una hoja de Excel en que se iban introduciendo y transcribiendo en los archivos de código. La mayor parte de estos fallos se producían por equivocaciones en el tamaño en bytes de ciertos tipos de datos ocasionando que dos valores compartiesen algunas direcciones de memoria. Aunque la terminal de depuración facilitó bastante el proceso, sin un auténtico depurador en línea seguía siendo muy difícil determinar si un determinado fallo se producía por un conflicto de direcciones, por un desbordamiento del watchdog o por un error lógico en las rutinas de proceso de cada dispositivo.

Otra gran fuente de errores fue la comprobación del funcionamiento de los primeros dispositivos hardware pues los cables Dupont[6] eran de mala calidad y hacían mal contacto multiplicando la incertidumbre cuando se producía algún fallo.

En el lado positivo, gracias a la rutina de reinicio de los puertos del shield de Ethernet W5100 se solucionó el problema de desconexión de la red zanjando uno de los fallos recurrentes de la placa. El watchdog también ayudó a acelerar considerablemente la detección de errores pues la controladora se reiniciaba al encontrar una situación de bloqueo en lugar de quedar esperando indefinidamente, lo que dejaba dudas de que pudiera fallar el servidor Web.

5.1.3. Desarrollo del shield Vesta A

La acumulación de errores, el funcionamiento deficiente del servidor HTML y los problemas con el hardware unidos a un pico de esfuerzo ante la preparación de los exámenes de Junio de 2019 aconsejaron detener el desarrollo para mejorar las herramientas de depuración y, de paso, calmar un poco la frustración ante tantos errores seguidos.

Tras los exámenes se dejó el desarrollo de software en suspenso pasando al diseño de una placa de circuito impreso como un shield de Arduino más preparado para manejar ocho entradas binarias optoacopladas[7], seis salidas amplificadas con transistores con potencia suficiente para

[6]Son clavijas en miniatura que sirven para conectar los terminales de los circuitos uno a uno. Pueden ser de tipo macho o hembra.

[7]El optoacoplamiento es una técnica que consiste en detectar ópticamente la presencia de corriente eléctrica mediante un led y una fotocélula, ambos contenidos en un encapsulado similar al de un circuito integrado. De

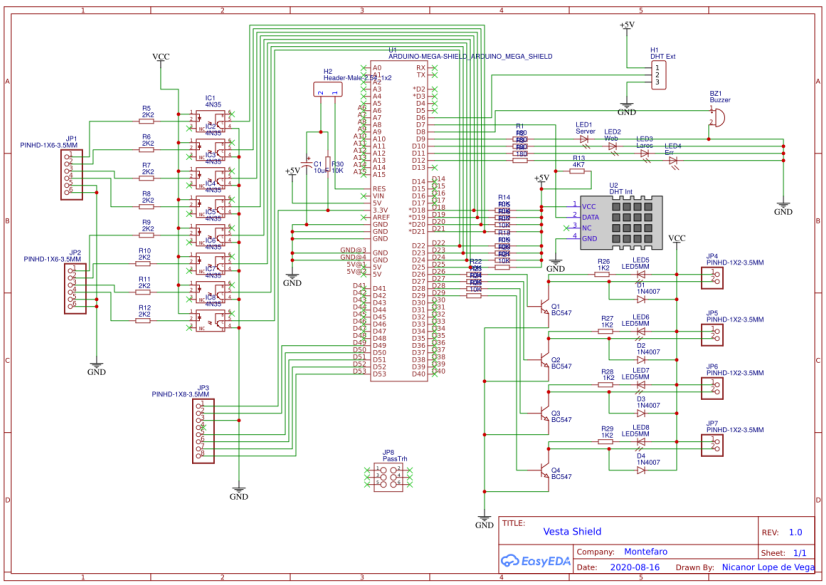


Figura 5.2: Esquema eléctrico del circuito Vesta A

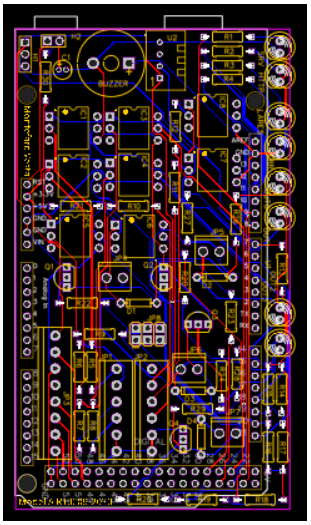


Figura 5.3: Trazado de pistas y diseño de Vesta A

encender pequeñas lámparas incandescentes o excitar relés industriales (figura 5.2).

Para facilitar la realización de la placa no teniendo que soldar puentes se usó un diseño de doble cara de pistas de cobre (figura 5.3). El resultado final se puede apreciar en la figura 5.4. Se añadió un sensor de humedad y temperatura DHT10[8], un zumbador piezoeléctrico y cuatro diodos led concebidos para mostrar el funcionamiento de los distintos componentes del kernel: Error, Web, Lares y Modbus[9].

esta forma se logra un aislamiento galvánico total entre el circuito digital del procesador y el de la entrada, que puede funcionar con otro voltaje y con otra masa

[8] https://www.es.co.th/Schemetic/PDF/ASAIR_DHT10.PDF

[9] Modbus es un protocolo de control industrial muy extendido en la automatización. Aunque el propósito de Vesta es servir al controlador domótico Lares, agregar un servidor Modbus permitía a Vesta hacer migraciones de sistemas existentes al nuevo concepto

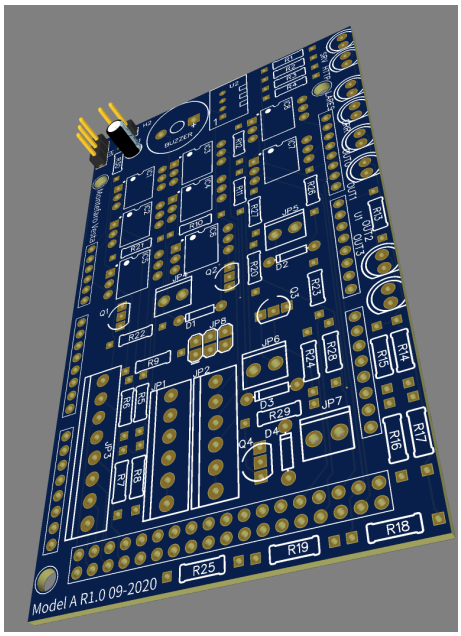


Figura 5.4: Infografía del shield Vesta A

Tras el diseño de la placa mediante la herramienta **EasyEda**[10] se encargó una pequeña serie de prototipos a un fabricante asiático. Cuando se ensamblaron las tres primeras unidades se comprobó que el funcionamiento era satisfactorio y al final de septiembre se reanudó el desarrollo de software.

5.1.4. Tercera iteración

El framework Bootstrap era muy potente. Quizá demasiado para la funcionalidad requerida de configurar una controladora de dispositivos. Como sería muy complejo albergar todas las librerías en el espacio limitado de un módulo Arduino, el archivo HTML lo cargaba siguiendo un enlace a la propia página de Boostrap. Esto quería decir que si la red domótica en que se instalaba el controlador no tenía conexión con internet no se cargaría el framework y la visualización de los datos sería caótica. Incluso satisfaciendo el injustificable requisito de tener conexión a internet, los parámetros de conexión (Ip, puerta de enlace y máscara de red) se configuran también desde el navegador lo que implica que al menos en la primera configuración era prácticamente improbable tener ese enlace a internet con los ajustes de fábrica.

Era imprescindible que los controles y estilos mostrados residiesen en la propia placa dejando las referencias para elementos secundarios prescindibles como dibujos, logotipos o archivos de ayuda. Se buscaron los frameworks más reducidos en internet decidiendo usar el **Min Framework** de Owen Veersteg[11] convenientemente modificado para el trabajo requerido. A pesar de lo reducido de su tamaño era inasumible almacenarlo en el microcontrolador como un array constante de caracteres. Se decidió desarrollar una rutina para convertir un archivo binario o de texto en un array de bytes que también se empleó para albergar iconos que identificasen los tipos de dispositivos.

[10] https://easyeda.com/es
[11] https://github.com/owenversteeg/min

Con estos cambios y a pesar del considerable aligeramiento de la clase del servidor HTML el compilador comenzó a dar errores por haber rebasado el espacio destinado a variables globales. En estas condiciones el desarrollo quedaba seriamente comprometido pues parecía haberse alcanzado el límite del modelo más potente de Arduino quedando todavía muchos más tipos de dispositivos y funcionalidad que incorporar.

Investigando se encontró una directiva del compilador de Arduino que permitía almacenar las constantes de texto y los vectores estáticos en la memoria del programa (considerablemente superior al espacio Ram disponible) a cambio de acceder a estos valores mediante unas funciones especiales. Se remodeló el servidor HTML completo trasladando todas sus constantes y el compilador dejó de dar error. A cambio, las funciones de acceso penalizaban la ejecución haciendo que el servidor fuese todavía más lento, provocando desbordamientos del watchdog lo que obligó a colocar llamadas a la función de reinicio del temporizador que ralentizaban todavía más el servicio y dejaban en suspenso las funciones de los dispositivos. El proyecto parecía haber entrado en otra fase crítica.

Pruebas

Si no se usaba la web para configurar los dispositivos el rendimiento de la controladora era óptimo. Superior en casi todo al de las placas controladoras que funcionaban en aquel momento en la casa, mucho menos asequibles. El servidor Modbus permitía que los prototipos Vesta pudieran entrar en producción con la instalación domótica existente. Como se trataba de seguir adelante, se instalaron dos de estos prototipos de Vesta A en uno de los dormitorios y en la zona de la comunidad de vecinos que controlaba la iluminación del garaje, la puerta motorizada y los pulsadores asociados. Este último controlador está separado de la red doméstica por los problemas con el servidor Web y sigue en producción en el momento de la redacción de este documento.

De nuevo surgieron errores relacionados con los cálculos de las direcciones de memoria de los dispositivos alimentando la incertidumbre ante la dificultad de depurar.

5.1.5. Cuarta y quinta iteraciones

Durante la parte más restrictiva de la pandemia de Covid-19 se intensificaron los esfuerzos para resolver los problemas con los controladores Vesta. El servidor HTML seguía siendo la principal fuente de problemas hasta el punto de plantear la tentación de cambiar la estrategia y crear un dispositivo hardware basado también en Arduino que haría las veces de terminal de depuración basado en el puerto serie que tan bien había estado funcionando para localizar los errores. La idea era conectar este aparato a los módulos para obtener diagnósticos y cambiar cualquier parámetro. Mientras tanto se añadían funciones al terminal de depuración existente y se eliminaban líneas de código del servidor en un vano intento por mejorar su rendimiento a base de arañar milisegundos. Sin embargo era muy difícil acelerar un servidor web que recargaba la página entera cada vez que respondía a un comando del usuario. Asumiendo que éste sólo debía acceder a las configuraciones con motivos de mantenimiento, se pensaba dejar como estaba el controlador y comenzar ya con el procesador central.

En estas dos iteraciones apareció un problema con el que no se había contado. La estructura de los registros de cada tipo de dispositivo, que habían sufrido modificaciones cada vez que se decidía incorporar un nuevo campo para ampliar la funcionalidad, debían ser replicados en las comunicaciones con el procesador central de Lares. En un principio se pensó en filtrar un subconjunto de estos datos para ahorrar en el tamaño de los paquetes Ip a transmitir, pero la tarea desbordó las previsiones alargando nuevamente los plazos y llegaron los exámenes de febrero del año 2020, paralizando nuevamente el desarrollo.

Pruebas

El servidor web resultaba inmanejable. A pesar de hacer lo que debía, los tiempos eran inasumibles. Aunque el funcionamiento del módulo era correcto, no se podía considerar el resultado como un producto válido asumiendo que debía ser la base de este concepto.

5.1.6. Desarrollo del shield Vesta A-Plus

Una vez terminadas las evaluaciones y con la moral más baja de todo el tiempo del proyecto, se decidió mejorar el diseño del shield Vesta-A que resultó tremendamente práctico, aunque limitado debido a que en el tamaño de un shield de Arduino apenas había espacio para la gran cantidad de componentes electrónicos necesarios para gestionar los dispositivos. Componentes pasivos como las resistencias ocupaban demasiado para la función que realizaban y aunque se intentó reposicionar los elementos de la primera versión dejando espacio libre, apenas cabía un 10 % más de lo que ésta contenía.

El desarrollo estaba totalmente detenido por cuestiones ajenas al proyecto. Se aprovechó esta pausa para experimentar con la tecnología SMD[12] probando antes con kits de iniciación. Se comprobó que los resultados eran óptimos y se abordó un nuevo diseño que incrementó considerablemente los componentes instalados (figura 5.5). La nueva Vesta A-Plus tiene 16 entradas binarias optoacopladas, 12 salidas amplificadas con transistor, un módulo DHT10, un puerto para un segundo DHT10 externo, zumbador, fuente de alimentación, cuatro teclas programables, puerto para módulo lector RFID y cinco leds de control. Igual que con el shield anterior, se utilizó un circuito impreso de doble cara con uniones entre pistas por orificios metalizados (figura 5.6). Para colocar correctamente la pasta de soldadura en las zonas indicadas se utilizó la técnica de stencil que consiste en colocar una placa de metal con orificios situados justamente donde hay que depositar la pasta. Este proceso que normalmente se realiza de forma automática también permite una aplicación manual. Se coloca el stencil sobre el circuito y se esparce el fundente presionándolo contra los orificios con una espátula. Al terminar se levanta el stencil y los pads quedan preparados para albergar los componentes, que se colocan con unas pinzas y después se aplica aire caliente para fundir el metal. El resultado es perfecto.

El diseño se realizó con la misma herramienta (figura 5.7), se encargó al mismo fabricante, aunque el pedido inicial fue de cinco unidades ante la posibilidad de minimizar las consecuencias de algún probable fallo de diseño en el prototipo. Las dos primeras unidades se completaron

[12]SMD o SMT son las siglas de Surface-Mount Design o Surface-Mount technology y consisten en posicionar los componentes sobre un sustrato de fundente que se calienta por convección. Con esta técnica aumenta el grado de precisión de posicionado y se miniaturiza considerablemente el circuito.

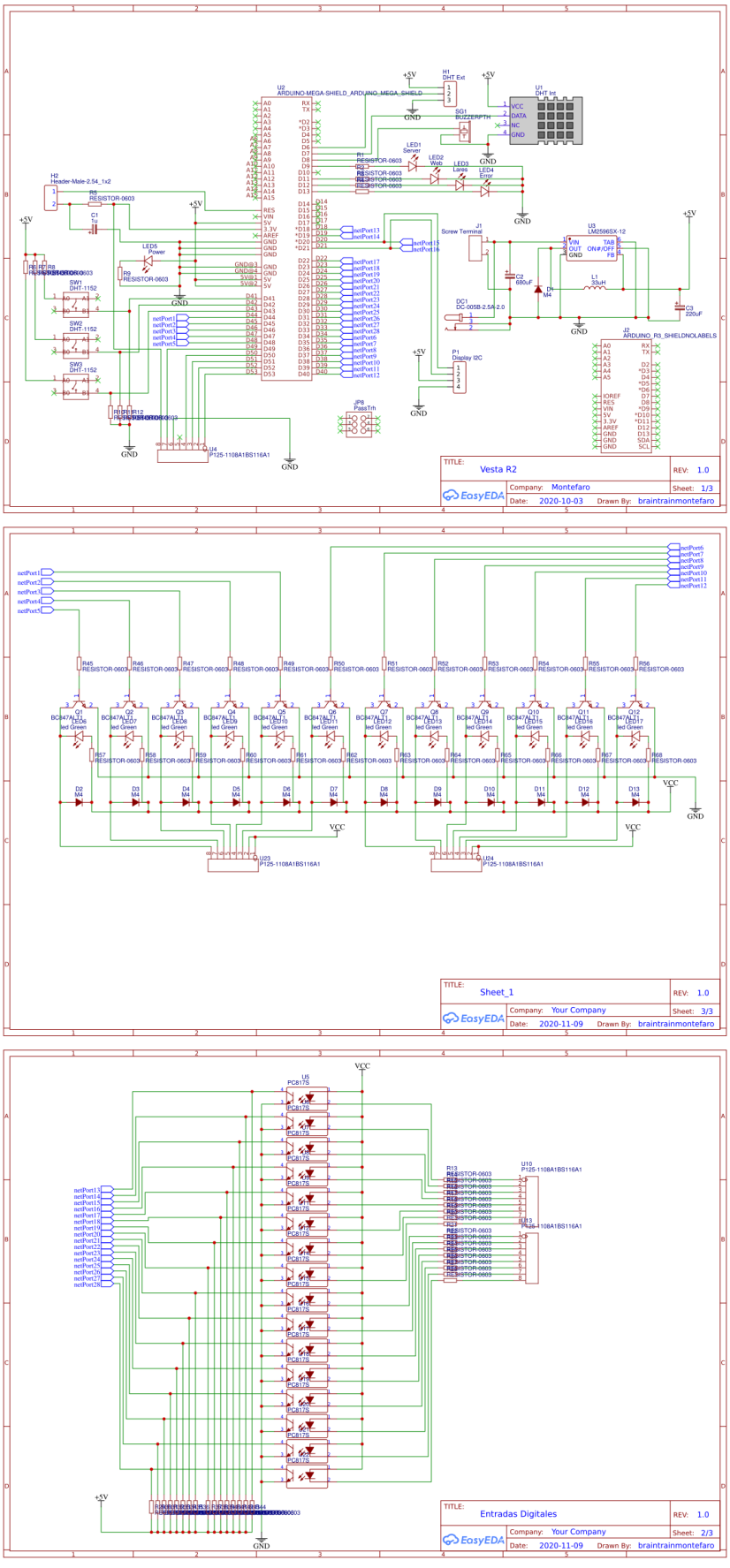


Figura 5.5: Esquema eléctrico general del circuito Vesta A Plus

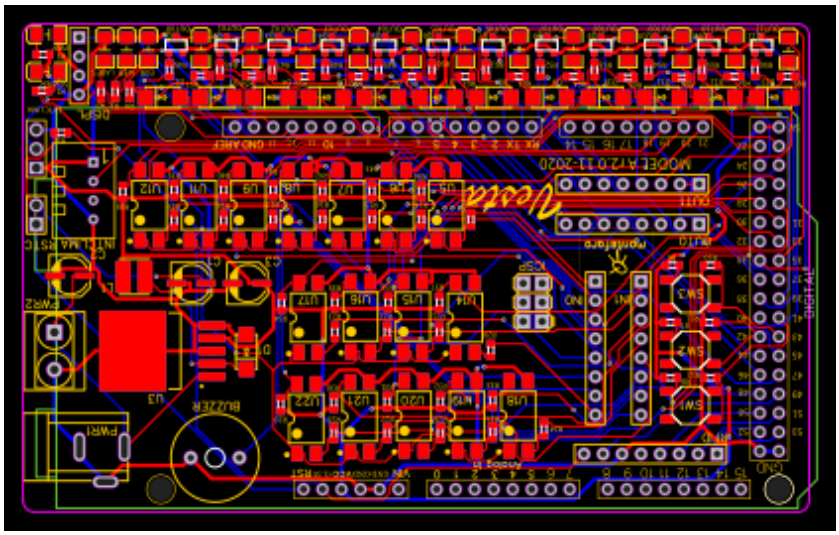


Figura 5.6: Trazado de pistas y diseño de Vesta A Plus

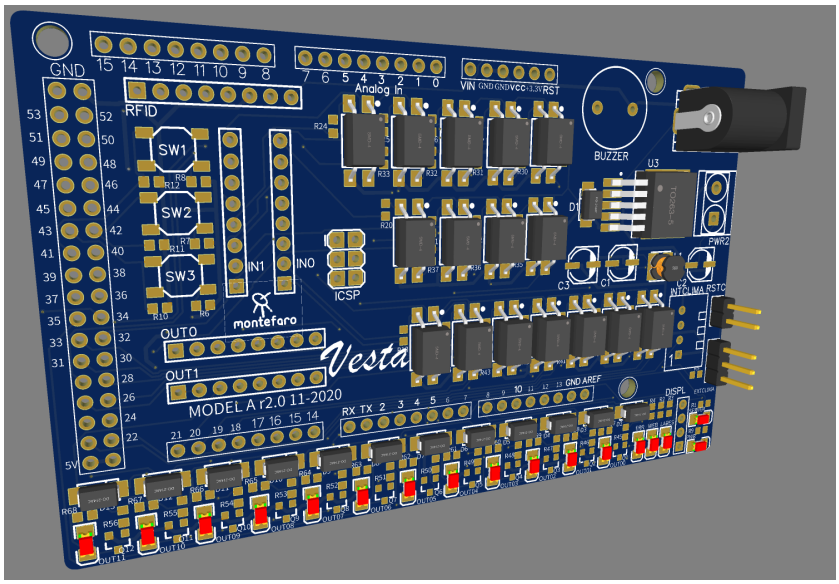


Figura 5.7: Infografía del shield Vesta A Plus

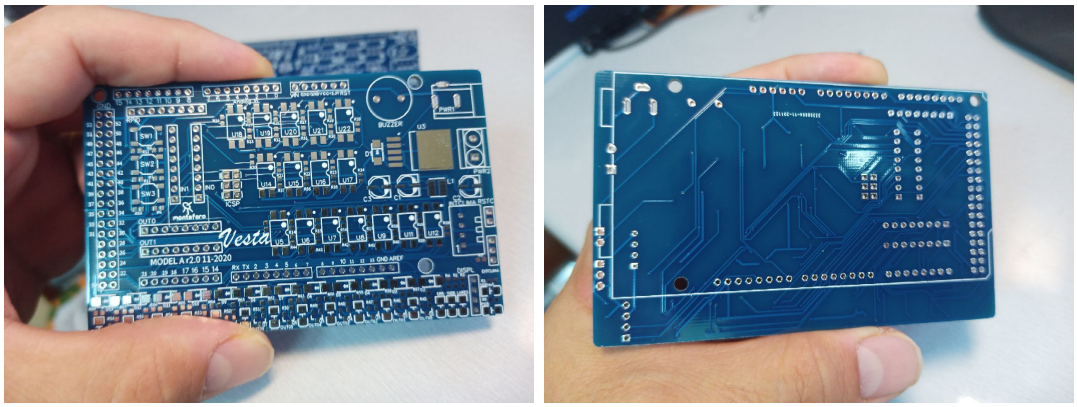


Figura 5.8: Anverso y reverso de la placa manufacturada (Vesta A Plus)

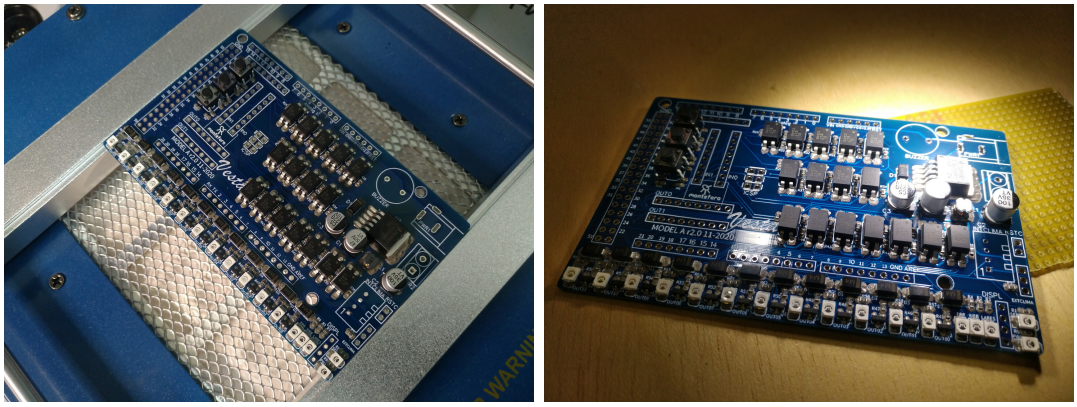


Figura 5.9: Proceso de soldadura SMD de componentes

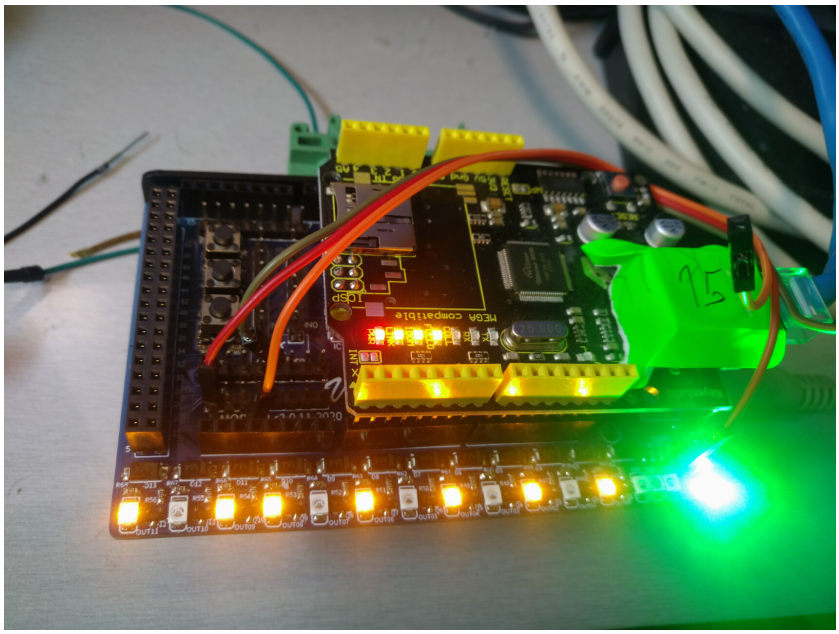


Figura 5.10: Prototipo en pruebas

nada más llegar las placas (figuras 5.8 y 5.9), una de ellas entró en producción en el armario eléctrico principal de la casa y la otra se asignó para continuar con las pruebas (figura 5.10).

5.1.7. Sexta iteración

Era imprescindible replantear todo lo construido hasta ese momento. Tras una fase totalmente improductiva del proyecto en el curso 2020-2021 con motivo de los estudios, se decidió acometer dos cambios muy importantes que implicarían retroceder conceptualmente hasta las primeras fases.

- **Compilador Vesta**: La programación de los dispositivos no era eficiente. La mayor parte del esfuerzo se dedicaba a calcular a mano las direcciones en memoria de los parámetros. Había que replantear todos los cálculos si se añadía algún nuevo campo lo que hizo que se crearan datos que no se usaban en las primeras versiones por si era necesario incorporar alguno más desperdiciando memoria con ello. Incluso con esta prevención surgía a menudo la necesidad de replantearlos repitiendo las cuentas y apareciendo nuevos errores. Si el 90 % del código fuente eran declaraciones de punteros basados en constantes había que encontrar una forma de generar todo de forma automática. Definiendo los datos en un lenguaje de alto nivel como C#, con los conocimientos recién adquiridos en las asignaturas **Procesadores del Lenguaje I** y **Procesadores del Lenguaje II**, se abría una nueva posibilidad para dejar que el computador hiciese de una vez todos los cálculos de referencias.
- **Servidor Web**: La mejor decisión del proyecto fue también la más drástica... Se abandonó todo el trabajo realizado comenzando de nuevo. Una parte del nuevo servidor Web se generaba también con el compilador, que ahora podía transcribir archivos enteros en variables estáticas alojadas en la memoria del programa. Ya no era necesario hacer nada a mano para alojar en Arduino el framework, las hojas de estilo CSS, el icono de carga de la página y, especialmente archivos de código JavaScript.

Fue esta nueva posibilidad la que abrió el camino al desarrollo de un programa a ejecutar desde el lado del cliente. El nuevo servidor realizaba peticiones sencillas al servidor. Éstas eran la estructura de la controladora o el contenido actual de cualquiera de las tres memorias. La estructura de la controladora es una cadena alfanumérica con una codificación sencilla que informa al programa del lado del cliente del contenido de la tabla de slots. Con esa información se genera una estructura de JScript que contiene toda la información para rellenar los controles de cada dispositivo o mostrar una vista general.

Con la información de la ubicación en memoria de cada campo o parámetro que forma parte del código fuente JavaScript y que ha sido generada por el compilador Vesta, el navegador accede directamente a los datos que necesita mapeando en la cadena que el servidor HTML proporciona a intervalos regulares el contenido de ese control. El proceso es tan ligero porque el servidor Web sólo necesita responder a las peticiones de contenido de la memoria que se elaboran con una simple iteración lineal sobre el vector solicitado.

Todo el proceso de componer la vista, colocar los datos y añadir las rutinas de respuesta a eventos en los controles las realiza el procesador del lado del cliente, mucho más rápido que el

propio Arduino y capaz de mostrar la información demandada de forma interactiva usando los datos del último envío que residen en la estructura del cliente. En el otro sentido la respuesta es también muy rápida porque sólo se envía el dato modificado, que la controladora puede procesar empleando el mismo tiempo que le lleva atender cualquier evento de los dispositivos de hardware.

Con esta nueva forma de funcionamiento sólo es relativamente lenta la carga de la página la primera vez, aunque se ha diseñado de forma que presenta ya la información a medida que se va componiendo eliminando la percepción en el usuario de congelación de la interface. Otra gran ventaja es que ampliando el número de tipos de dispositivo o el número de dispositivos instalados en la controladora se penaliza mínimamente la comunicación resultando altamente escalable.

Esta nueva fase del desarrollo fue más lenta durante la creación del compilador. En ese proceso no se tenía la certeza de que el código ahora generado por el compilador fuese más eficiente. Además se decidió incorporar ciertas optimizaciones, como la inclusión de archivos fuente en C++ con el código de servicio del núcleo y de los dispositivos, que al principio iban a ser generados por el mismo compilador introduciendo línea a línea el código C++ en llamadas de código C#.

Pruebas

Las primeras compilaciones arrojaron numerosos errores sintácticos que costó corregir casi la mitad del tiempo que llevó crear el propio compilador. Y cuando se resolvieron apareció un nuevo módulo Vesta totalmente funcional, increíblemente más rápido que el anterior, sin fallos de asignación de memoria, preparado para encajar en cuestión de horas cualquier modificación profunda en el funcionamiento de cualquier dispositivo.

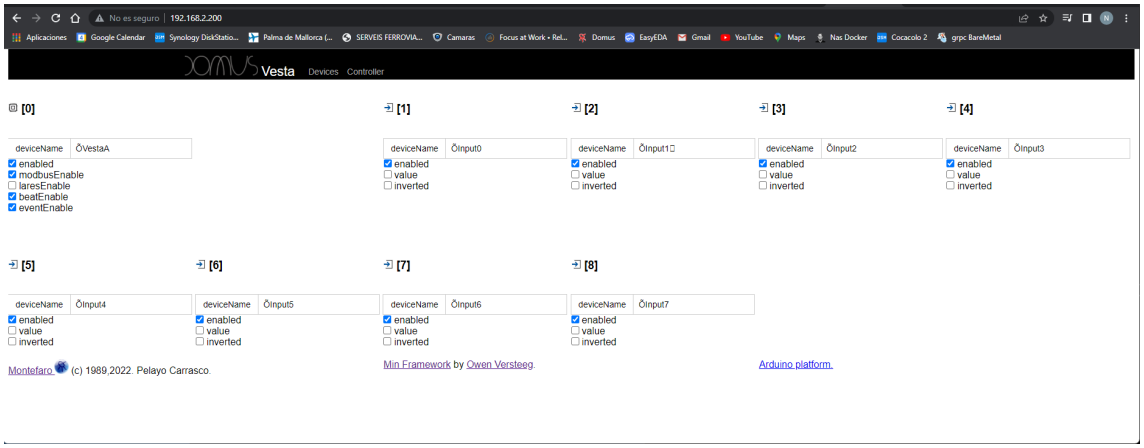


Figura 5.11: Web Vesta con framework Min (Primer ensayo)

El nuevo servidor web era rápido y eficiente. Quizá no tan vistoso como el de Bootstrap, aunque la rapidez de reacción lo compensaba (figura 5.11). Se podían conectar varios equipos a la misma controladora y ésta era capaz de servir información a todos ellos manteniendo el funcionamiento de los dispositivos hardware sin acusar problemas de rendimiento.

5.1.8. Séptima iteración

Dada la potencia del servidor Web se decidió ampliar las prestaciones extendiendo la terminal de depuración operativa en el puerto serie a la página de configuración. De esta forma ya no sería necesario conectar nada en el puerto serie de una controladora en producción y se podría llevar toda la funcionalidad ampliando las posibilidades todavía más.

Se crearon nuevos tipos de dispositivo: Entrada y salida analógicas, puertas lógicas para combinar las salidas de varios dispositivos, sensores de umbral para convertir en binaria una entrada analógica.

Se amplió la funcionalidad de los dispositivos existentes añadiendo nuevos parámetros de configuración.

Pruebas

Aparte de los inevitables pequeños errores fáciles de corregir, parecía existir un límite en el número de dispositivos instalados en la controladora a partir del cual se producían fallos de funcionamiento que apuntaban a posiciones de memoria corruptas. Calculando el espacio ocupado en memoria por la estructura no parecía tener lógica; era bastante inferior al valor teórico de 8Kb direccionado por el procesador. Tras repetir los ensayos con diferentes configuraciones de dispositivos se determinó que el problema ocurría al superar cierto número de dispositivos (de 20 a 24 dependiendo del tipo de dispositivos de la configuración, pues éstos ocupan diferentes tamaños en memoria).

En todos los casos, al superar el límite, la tarjeta arrancaba correctamente y tras unos segundos se reiniciaba incluso desactivando el perro guardián o continuaba en funcionamiento, pero no contestaba a las peticiones por el puerto HTTP. Se hacía necesario comprender la naturaleza del error y si éste se producía por un consumo excesivo de memoria había que encontrar una fórmula que permitiese calcular el espacio real ocupado para que el compilador pudiera emitir mensajes de advertencia si se forzaba la configuración.

Revisando el código se comprobó que la versión en pruebas generaba una instancia de la clase del dispositivo por cada uno de ellos acumulando en memoria un espacio al que había que sumar el que ya se reservaba para sus datos. La instancia del dispositivo contenía variables privadas o protegidas que se utilizaban para cálculos internos, normalmente relacionados con los temporizadores (el intervalo entre muestreos del termostato, el antirrebotes en las entradas y el de apagado o encendido en las salidas), que no se estaban añadiendo a la cuenta. Se determinó que estas variables se usaban a menudo para hacer cálculos que afectaban a todas las instancias de la misma clase de dispositivo e incluso degradaban la eficiencia del sistema porque en los cálculos cuyo resultado afectaba a todos los dispositivos del mismo tipo, se repetía para cada uno.

Era evidente que el código requería una modificación bastante importante.

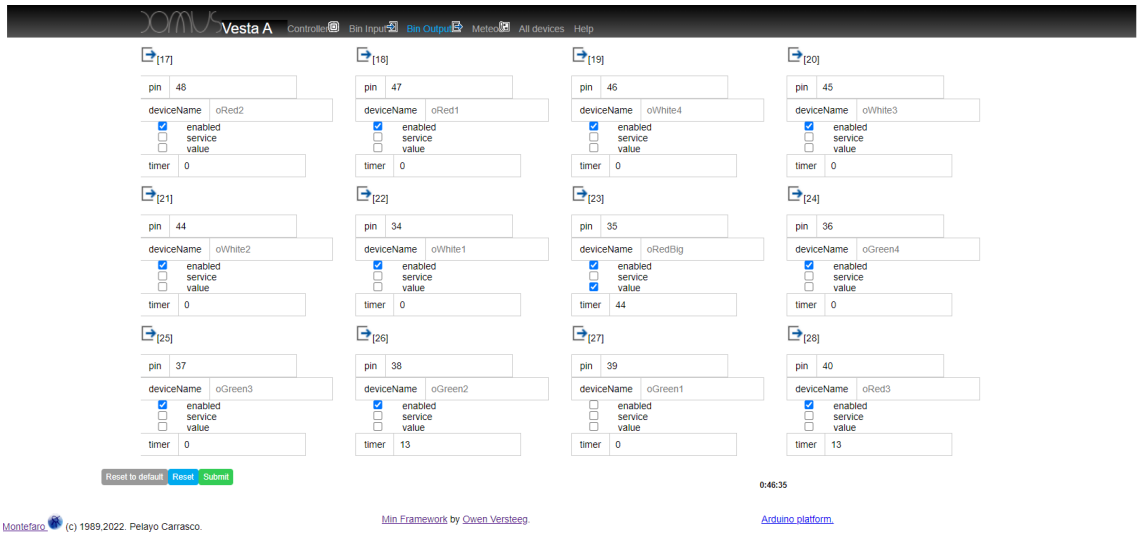


Figura 5.12: Web mejorada. Se añaden iconos de dispositivos.

5.1.9. Octava iteración

El nuevo rediseño consistía en cambiar el nombre de las clase base de los dispositivos que pasó de ser **vestaItem** a **vestaDeviceType**. Se creó un vector de tipos instalados que funciona como un diccionario que traduce el valor del tipo de cada dispositivo a un valor ordenado de 0 al número de dispositivos diferentes menos uno. Es decir; si se han instalado los tipos 0, 3, 9 y 100, el diccionario asigna a 0 el valor 0, al 3 el valor 1, al 9 el valor 2 y al 100 el valor 3. La clave del diccionario es el índice de cada elemento en el vector. Se añadió una tabla de instancias en una nueva clase llamada *distroItems* generada por el compilador en la que se crea una única instancia de cada tipo de dispositivo y se modificó la tabla de slots de forma que ahora los dispositivos hacen referencia al número de tipo guardado en esta tabla de instancias. Se añadió en el kernel una función *getDevice(x)* que accede a la referencia del tipo de dispositivo usando la tabla de slots y asigna los valores de offset que apuntan a las tres memorias para esa instancia concreta. Como varios accesos a un mismo dispositivo pueden ir seguidos, se añadió una variable que contiene la última referencia asignada que hace la función de caché para agilizar lo posible esta función, que se prevé crítica.

Se añadió una nueva función de bucle por tipo de dispositivo en la que el kernel invoca los cálculos que afectan a todas las instancias y se revisó cada tipo de dispositivo trasladando las variables protegidas correspondientes a cada instancia a nuevos valores reservados en la memoria RAM administrada o llevando el código común a la nueva función por clase.

Adicionalmente se hicieron mejoras menores en el servidor HTML (figura 5.12), se añadieron funciones al terminal de depuración permitiendo habilitar o deshabilitar dispositivos con un comando de texto, se creó un detector de interbloqueos basado en los últimos n-accesos al kernel, deshabilitando la instancia que haya ocupado n-llamadas seguidas, se modificó la rutina de bucle principal utilizando el flag de habilitación del dispositivo kernel para activar o desactivar el bucle de funcionamiento y se añadió una función especial de recuperación de errores basada en un pin de entrada del microcontrolador desactivando la ejecución del bucle principal si esta entrada está conectada con el valor *gnd*. La idea es asociar la entrada de inhabilitación con un pulsador de *reset* en el diseño de la próxima shield Vesta.

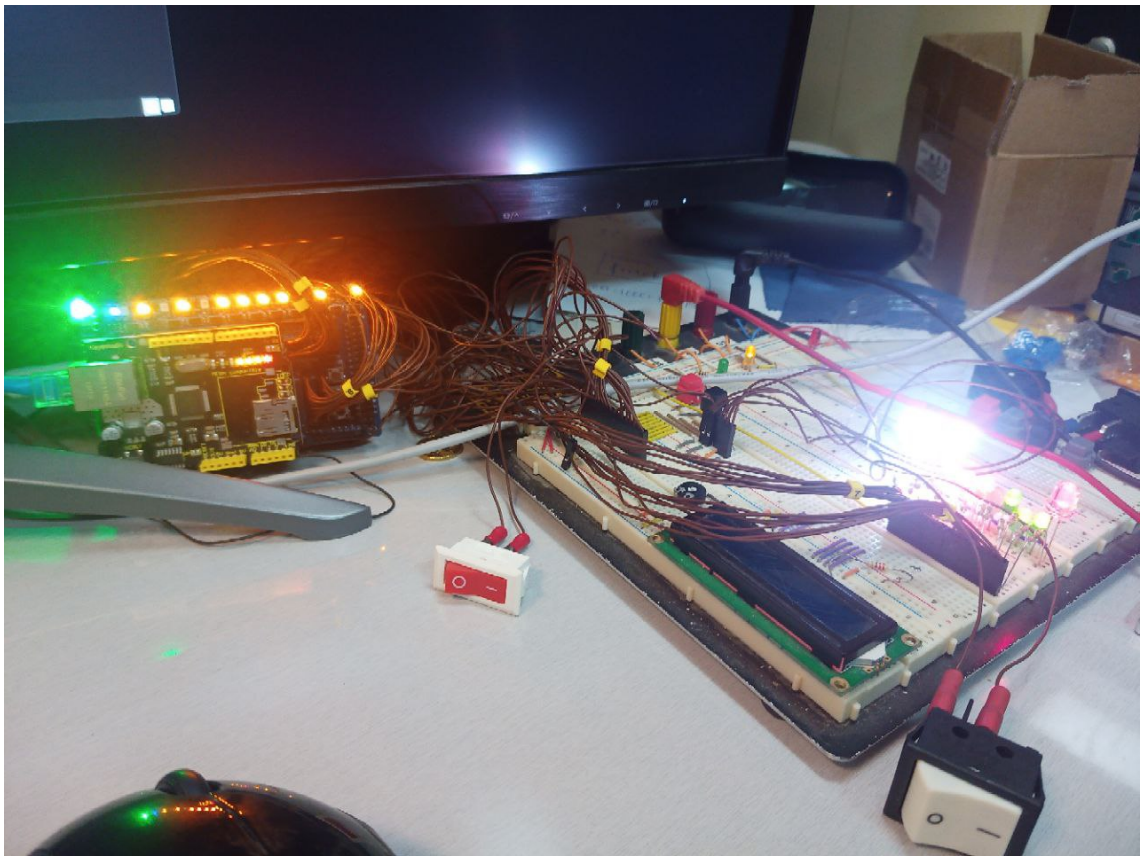


Figura 5.13: Banco de pruebas para entradas y salidas binarias con Vesta A Plus

Pruebas finales

Tras lidiar con los numerosos errores típicos de un cambio tan profundo como el realizado, se comprobó que ahora el funcionamiento es más fiable, la arquitectura más robusta y que el límite de dispositivos instalados sin fallos de corrupción de memoria se ha elevado a 28 o 30. De las pruebas de funcionamiento se ha pasado a un testeo intensivo en banco (figura 5.13), siempre en modo local, asignando diferentes modos de funcionamiento a los componentes instalados y dejándolos en marcha durante largos períodos de tiempo.

En todos los casos el comportamiento de la controladora ha sido el esperado, concluyendo que el producto final es apto para la función que lo describe en este trabajo.

5.2. Puntos a mejorar

Un proyecto tecnológico nunca está totalmente listo. Con el uso siempre se descubren posibles optimizaciones, nuevos dispositivos, mejoras de interface... El controlador Vesta necesita evolucionar hacia una arquitectura más potente como el procesador ESP32 o incluso un chip ATMEL en un circuito dedicado. Al haber desarrollado el compilador que genera el código fuente c++ de Arduino, es posible aprovechar la generación automática de éste para crearlo en otro lenguaje de alto nivel propio del procesador de destino o incluso en el código máquina optimizado del que se elija. Incluso es posible ir más allá; Vesta puede convertirse en un concepto independiente

de la arquitectura hardware, encapsulado en distintos procesadores de acuerdo con la función que deba realizar en el hogar, estableciendo gamas por prestaciones. Volviendo a la versión de Arduino actual, falta invocar al compilador nativo para cerrar en un solo paso la programación de la placa, de la misma forma que hace el plugin de Visual Studio Code.

Es necesario retocar la presentación de la web de Vesta haciéndola más agradable a la vista, aprovechando más el framework. Se debe rellenar la documentación de ayuda, que actualmente está disponible mediante un enlace. Pensando en el funcionamiento sin conexión a internet habrá que trasladar a la memoria flash de programa los puntos que sea necesario tener offline. También habrá que mejorar el terminal añadiendo funciones, especialmente en los modos de servicio, para que el instalador domótico pueda verificar el funcionamiento, realizar todo tipo de pruebas y ajustes sin necesidad de herramientas de configuración externas.

Sin duda la prueba exhaustiva del sistema en producción que se producirá tras la terminación de este proyecto arrojará nuevas mejoras que darán lugar a más iteraciones evolucionando este concepto durante muchos años.

5.3. Compilador para firmware placas Vesta

La base del funcionamiento del controlador domótico Vesta es una estructura de información que corresponde a todos los dispositivos instalados repartidos en los tres tipos de memoria. Cada instancia tiene acceso a unas direcciones concretas que calcula en tiempo de ejecución sumando a la dirección inicial un valor de offset que corresponde a cada parámetro. La gestión a mano de estas direcciones es laboriosa y propensa a errores de cálculo que dan lugar a datos corruptos y los errores de funcionamiento que éstos conllevan. Por ello se decidió crear un compilador que generase el código fuente c++ de Arduino calculando la memoria y generando todas las clases relacionadas con el acceso a memoria, que son el almacén, el servidor web y las de cada tipo de dispositivo.

El compilador procesa dos objetos C# llamados **Version** y **Distribution**.

En el archivo de versión se enumeran los distintos tipos de dispositivo presentes en esta generación de controladores. Es importante tener en cuenta que los tipos de dispositivos que sirven como parámetro en las llamadas *addDevice* corresponden con una versión concreta de ese tipo de dispositivo. La clase *BInput* referida en este contexto podría ser una versión más actual que la contenida en otra versión de Vesta aunque mantenga una funcionalidad equivalente. La filosofía del diseño permite que puedan coexistir diferentes versiones del mismo tipo de dispositivo en distintas controladoras y, aunque no aportaría ninguna ventaja a primera vista, en la misma, apareciendo dos veces en el diccionario de tipos, con dos identificadores distintos.

```
public class Version3:VersionBase
{
    public Version3():base()
    {
        addDevice(new Controller("controller", "Onboard controller"));
        addDevice(new BInput("bInput", "Binary input"));
        addDevice(new BOutput("bOutput", "Binary output"));
        addDevice(new DHT("dhtVesta", "Meteo module"));
        addDevice(new logicGate("logicGate", "Logic gate"));
        addDevice(new trigger("trigger", "Analogic trigger"));
```

```
            addDevice(new pulseCounter("pulseCounter", "Pulse counter"));
        }
    }
```

El código de la distribución es una definición de valores seguida de una enumeración de las instancias concretas que pertenecen a los diferentes tipos de la distribución en la que se basa, de una forma similar al modo descriptivo de **VHDL**[13]

En la primera parte se definen variables que el compilador necesita:

- **VName**: Nombre de la versión. Este texto da nombre a la clase principal, al archivo Arduino (extensión .ino) y aparece en encabezado de la barra de herramientas de la web de configuración.
- **VComments**: Notas sobre la versión. Aparece en los comentarios del código objeto, en la ayuda de la consola de comandos del puerto serie y también en este puerto cada vez que se inicia la controladora.
- **VHardwareVersion, VHardwareRevision**: Valores numéricos correspondientes a la versión de la controladora. Estos valores son utilizados por el procesador domótico Lares para establecer el protocolo de transmisión y las capacidades que puede exigir de la controladora.

Además de estos valores, el compilador necesita un parámetro de 32 bits almacenado en dos enteros sin signo de 16 que contiene el número de serie. Este identificador no aparece como constante en el código fuente porque es el programa principal el encargado de suministrarlo. De momento está almacenado en una constante, pero está previsto que se lea de una base de datos en la que se dan de alta las licencias vigentes y los clientes a las que pertenecen.

La función **createInstances** añade los dispositivos que contendrá el controlador menos el propio dispositivo controlador que siempre se añade en el slot 0. El orden de inclusión con la función **addInstance** define el número de slot que tendrá cada dispositivo en la distribución. Antes de invocar a la función **addInstance** es necesario asignar un valor a la referencia **deviceBase** escogiendo uno de los dispositivos presentes en la distribución (código fuente anterior).

La función **populateInstances** es opcional. Asigna los valores por defecto que tendrán las propiedades seleccionadas de cada dispositivo. La llamada a la función **setDefaultValue()** de cada dispositivo acepta dos parámetros que son el nombre de una clave y un valor que puede ser de cualquier tipo. El compilador almacena estos valores en un diccionario y si encuentra alguna clave presente entre las propiedades de ese dispositivo, generará en su respectiva posición de memoria ese valor. En la memoria ROM ésta es la única forma de escribir, en la EEPROM los valores por defecto se escriben en un vector que se copia en EEPROM cuando el usuario invoca el comando de retroceder a ajustes de fábrica y en la RAM se escribe en una zona de inicio que se copia en RAM cada vez que la controladora se inicia.

```
namespace ArduinoCompiler.Vesta.Instances
```

[13]VHDL es un lenguaje de especificación definido por el IEEE siguiendo la norma ANSI/IEEE 1076-1993, utilizado para describir circuitos digitales y para la automatización de diseño electrónico. A estos lenguajes se les suele llamar **lenguajes de descripción de hardware** https://es.wikipedia.org/wiki/VHDL

```
{
    //Placa Vesta de tipo A.
    public class VestaA:Distribution
    {
        private Version2 mvarVersion=null;
        internal List<DeviceInstance> mcolInputs;
        internal List<DeviceInstance> mcolOutputs;
        public override string VName => "Vesta A";
        public override string VComments => "Standard Vesta device";
        public override byte VHardwareVersion => 4;
        public override byte VHardwareRevision => 0;

        public override VersionBase compilerVersion { get => mvarVersion; }

        public VestaA(UInt16 serial1, UInt16 serial0):base(serial1,serial0)
        {
            mvarVersion = new Version2();
        }
        protected override void createInstances()
        {
            DeviceBase deviceBase;
            DeviceInstance instance;
            base.createInstances();
            deviceBase = compilerVersion.getDeviceClass("bInput");
            for(int i = 0; i< 8;i++)
            {
                addInstance(deviceBase, string.Format("input{0:00}", i), (byte)(i + 22));
            }
            deviceBase = compilerVersion.getDeviceClass("bOutput");
            //Vesta A   Arduino
            addInstance(deviceBase, "output00", 48);
            addInstance(deviceBase, "output01", 47);
            addInstance(deviceBase, "output02", 46);
            addInstance(deviceBase, "output03", 45);
            addInstance(deviceBase, "output04", 44);
            addInstance(deviceBase, "output05", 34);
            addInstance(deviceBase, "output06", 35);
            addInstance(deviceBase, "output07", 36);
            addInstance(deviceBase, "output08", 37);
            addInstance(deviceBase, "output09", 38);
            addInstance(deviceBase, "output10", 39);
            addInstance(deviceBase, "output11", 40);
            deviceBase = compilerVersion.getDeviceClass("dhtVesta");
            addInstance(deviceBase, "meteoI", 7); //DHT interno
            addInstance(deviceBase, "meteoE", 6); //DHT externo
        }


        protected override void populateInstances()
        {
            base.populateInstances();
            //Creación de las instancias de los dispositivos contenidos
            //ROM
            controller.setDefaultValue("buzzer", (byte)8);
            controller.setDefaultValue("serverLed", (byte)9);
            controller.setDefaultValue("webLed", (byte)11);
            controller.setDefaultValue("tcpLed", (byte)11);
            controller.setDefaultValue("modbusLed", (byte)11);
            controller.setDefaultValue("errLed", (byte)12);
            controller.setDefaultValue("eventLed", (byte)255);
            controller.setDefaultValue("iddenId", (byte)10);

            //EEPROM
            controller.setDefaultValue("deviceName", "VestaA");
            controller.setDefaultValue("soundEnable", true);
            controller.setDefaultValue("watchdogEnable", true);
            controller.setDefaultValue("ethernetResetEnable", true);
            controller.setDefaultValue("webEnable", true);
            controller.setDefaultValue("modbusEnable", false);
            controller.setDefaultValue("laresEnable", true);
            controller.setDefaultValue("eventEnable", true);
            controller.setDefaultValue("beatTime", (UInt16)130);
            controller.setDefaultValue("ethernetResetTime", (UInt16)1000);
```

```
            //RAM
            controller.setDefaultValue("eventPort", (UInt16)5168);
            controller.setDefaultValue("webPort", (UInt16)80);
            controller.setDefaultValue("modbusPort", (UInt16)502);
            controller.setDefaultValue("laresPort", (UInt16)5169);
            controller.setDefaultValue("beatPort", (UInt16)5148);
            controller.setDefaultValue("localIp",
                new System.Net.IPAddress(new byte[]{192,168,2,200}));
            controller.setDefaultValue("mask",
                new System.Net.IPAddress(new byte[] {255,255,255,0 }));
            controller.setDefaultValue("gateway",
                new System.Net.IPAddress(new byte[] { 192, 168, 0, 1 }));
        }
    }
}
```

La ejecución del compilador produce la siguiente salida en pantalla:

```
Montefaro software (c) 1989 2022
(1/13) Copy sources...
(2/13) Build JS Controls...
(3/13) Copy debug web resources...
(4/13) Copy libraries...
(5/15) Build Cpp...
(6/16) Build Storage...
(7/13) Build devicesContainer...
(8/13) Build Web server...
(9/13) Build Kernel module...
(10/13) Build Ino...
(11/13) Creating implementation cart...
(12/13) Generating CS code for devices...
Done!
```

El aspecto del código generado es el siguiente[14]:

5.3.1. Clase de almacenamiento vestaConfig (header)

```
//Vesta compiler V1.0 28/06/2022 (c) 1989,2022

#include "vestaDeviceType.h"
#include "vestaKernelDef.h"

#ifndef vestaConfig_h
#define vestaConfig_h
#define MAX_RO 354
#define MAX_RA 206
#define MAX_EEPRO 693

class vestaConfig
{
public:
//Device data
static const unsigned int colDevices[MAXDEVICES][4]; //Device descriptor
static vestaDeviceType* colTypes[MAXTYPES]; //Device type register
static vestaDeviceType* deviceRegisterCache; //Last accessed device type
//Shared structures:
static unsigned char RA   [MAX_RA];
//RO: Read only default data
static const PROGMEM unsigned char RO [];
//MAC Address:
```

[14]Se ha eliminado parte del contenido no relevante para facilitar la comprensión

```
static const unsigned char defHDR[];
static const unsigned int defHDRCount = 204;
private:
static unsigned char mvarLastDeviceIndex; //Variable for registering types.
//EEPRO: Flash default data
static const PROGMEM unsigned char defEEPRO[];
static const PROGMEM unsigned char defRA[];
public:
                          vestaConfig () ; //Structure setup
static  void              assignTypeInstance (vestaDeviceType* rhs) ;
    //Assign external instance
static  vestaDeviceType*  getDevice (unsigned char slotId) ;
    //Return any vesta Item correctly fetched
static  void              debugRA () ;
    //Instant snapshot of RAM memory
static  void              debugRO () ;
    //Instant snapshot of ROM memory
static  void              debugEEPRO () ;
    //Instant snapshot of EEPROM memory
static  void              setDefaults () ;
    //All defaults
static  void              setDefaults (unsigned int offset, unsigned int size) ;
    //Custom defaults
static  void              setRaDefaults () ;
    //All defaults
static  void              setRaDefaults (unsigned int offset, unsigned int size) ;
    //Custom defaults
static  void              eeDump (unsigned int start, unsigned int amount) ;
    //EEPROM Memory dump
static  void              raDump (unsigned int start, unsigned int amount) ;
    //RAM Memory dump
static  void              roDump (unsigned int start, unsigned int amount) ;
    //ROM Memory dump
static  void              showPre () ;
    //Device board info.
static  unsigned char     peek (unsigned int address) ;
    //Read EEPROM address
static  void              poke (unsigned int address, unsigned char value) ;
    //Modify EEPROM address
};
#endif
```

5.3.2. Clase de almacenamiento vestaConfig (implementación)

```
//Vesta compiler V1.0 28/06/2022 (c) 1989,2022

#include "vestaConfig.h"
//Device type instances
const unsigned int vestaConfig::colDevices[MAXDEVICES][4]=
{
{0x0,0x0,0x0,0x0},//VestaA(controller)
{0x1,0x30,0x3F,0x28},//DVC001(bInput)
{0x1,0x3B,0x51,0x2C},//DVC002(bInput)
    //...contenido suprimido...//
{0x4,0x12D,0x239,0xAE},//DVC024(aInput)
{0x6,0x138,0x257,0xBC},//DVC025(logicGate)
{0x6,0x142,0x26D,0xBE},//DVC026(logicGate)
{0x5,0x14C,0x283,0xC0},//DVC027(aOutput)
{0x5,0x157,0x29C,0xC7},//DVC028(aOutput)
};
vestaDeviceType* vestaConfig::colTypes[];
vestaDeviceType* vestaConfig::deviceRegisterCache;
unsigned char vestaConfig::mvarLastDeviceIndex;
unsigned char vestaConfig::RA[];
const unsigned char vestaConfig::RO[]=
{
0x00,0x02,0x01,0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x56,0x65,0x73,0x74,
0x61,0x20,0x41,0x20,0x70,0x6C,0x75,
```

```
0x73,0x00,0x00,//0000
0x00,0x00,0x00,0x01,0x00,0x00,0x18,
0x42,0x1D,0x93,0x36,0x10,0x06,0x16,
0x09,0x0B,0x0C,0x08,0x0B,0x0B,0xFF,
0xFF,0xFF,0xFF,//0024
0x01,0x01,0x00,0x01,0x00,0x30,0x00,
0x28,0x00,0x3F,0x16,0x01,0x01,0x00,
0x02,0x00,0x3B,0x00,0x2C,0x00,0x51,
0x17,0x01,0x01,//0048

//...contenido suprimido...//

0x74,0x01,0xCC,0x28,0x03,0x01,0x00,
0x15,0x01,0x0C,0x00,0x78,0x01,0xE3,
0x07,0x03,0x01,0x00,0x16,0x01,0x17,
0x00,0x8C,0x01,//0264
0xFF,0x06,0x04,0x01,0x00,0x17,0x01,
0x22,0x00,0xA0,0x02,0x1B,0x00,0x04,
0x01,0x00,0x18,0x01,0x2D,0x00,0xAE,
0x02,0x39,0x01,//0288
0xA0,0x01,0x00,0x19,0x01,0x38,0x00,
0xBC,0x02,0x57,0xA0,0x01,0x00,0x1A,
0x01,0x42,0x00,0xBE,0x02,0x6D,0x05,
0x01,0x00,0x1B,//0312
0x01,0x4C,0x00,0xC0,0x02,0x83,0x04,
0x05,0x01,0x00,0x1C,0x01,0x57,0x00,
0xC7,0x02,0x9C,0x05
};
const unsigned char vestaConfig::defEEPRO[]=
{
0x00,0x56,0x65,0x73,0x74,0x61,0x41,
0x00,0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0xAF,0x01,0x00,0x82,
0x03,0xE8,0x00,//0000
0x14,0x14,0x30,0x00,0x50,0x01,0xF6,
0x14,0x31,0x14,0x1C,0xC0,0xA8,0x02,
0xC8,0xFF,0xFF,0xFF,0x00,0xC0,0xA8,
0x00,0x01,0x00,//0024
0x00,0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,0x44,0x56,0x43,0x30,0x30,
0x31,0x00,0x00,//0048
0x00,0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0x04,0x00,0x44,0x56,0x43,0x30,
0x30,0x32,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,0x00,//0072
0x00,0x00,0x04,0x00,0x44,0x56,0x43,
0x30,0x30,0x33,0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,0x04,
0x00,0x44,0x56,//0096
0x43,0x30,0x30,0x34,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,0x00,
0x04,0x00,0x44,0x56,0x43,0x30,0x30,
0x35,0x00,0x00,//0120

//...contenido suprimido...//

0x44,0x56,0x43,0x30,0x32,0x35,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,0x08,0x00,0x00,0x00,0x00,
0x00,0x44,0x56,//0600
0x43,0x30,0x32,0x36,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,0x00,
0x08,0x00,0x00,0x00,0x00,0x00,0x44,
0x56,0x43,0x30,//0624
0x32,0x37,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x27,0x10,0xFF,0x00,
0x44,0x56,0x43,//0648
0x30,0x32,0x38,0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,0x27,0x10,0xFF
};
```

```
const unsigned char vestaConfig::defRA[]=
{
0x04,0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,0x00,//0000
0x00,0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,0x04,0x00,0x00,0x00,0x04,
0x00,0x00,0x00,//0024
0x04,0x00,0x00,0x00,0x04,0x00,0x00,
0x00,0x04,0x00,0x00,0x00,0x04,0x00,
0x00,0x00,0x04,0x00,0x00,0x00,0x04,
0x00,0x00,0x00,//0048

//...contenido suprimido...//

0xC4,0x00,0x00,0x00,0x00,0x00,0x04,
0x00,0x00,0x17,0x70,0xF0,0x60,0x09,
0xC4,0x00,0x00,0x00,0x00,0x00,0x04,
0x00,0x04,0x00,//0168
0x04,0x00,0x00,0x00,0x00,0x00,0x00,
0x04,0x00,0x00,0x00,0x00,0x00,0x00
};
const unsigned char vestaConfig::defHDR[]={
0x1d
,0x00,0x0,0x0,0x0,0x0,0x0,0x0
,0x01,0x0,0x30,0x0,0x3F,0x0,0x28
,0x01,0x0,0x3B,0x0,0x51,0x0,0x2C
,0x01,0x0,0x46,0x0,0x63,0x0,0x30

//...contenido suprimido...//

,0x04,0x1,0x2D,0x2,0x39,0x0,0xAE
,0xA0,0x1,0x38,0x2,0x57,0x0,0xBC
,0xA0,0x1,0x42,0x2,0x6D,0x0,0xBE
,0x05,0x1,0x4C,0x2,0x83,0x0,0xC0
,0x05,0x1,0x57,0x2,0x9C,0x0,0xC7
};

//Structure setup

//...contenido suprimido...//



//Device board info.
void vestaConfig::showPre ()
{
vestaBasic::pps(F("Montefaro Hardware (c) 1989, 2022"));
vestaBasic::pps(F("Vesta model 'Vesta A plus' V4.0 (28-06-22)"));
vestaBasic::pps(F("Prototype with analog devices"));
vestaBasic::pps(F("Serial id 00000000-00000001"));
vestaBasic::pps(F("MacAddress: 18:42:1D:93:36:10"));
vestaBasic::pps(F("354 bytes of ROM, 206 bytes of RAM and 693 bytes of EEPROM."));
}
```

Todos los valores literales proceden de los cálculos realizados por el compilador y son aplicables exclusivamente a la unidad Vesta concreta que se ha compilado.

5.3.3. Ejemplo de la clase de un dispositivo generada por el compilador (implementación)

```
char* bOutput::getDeviceName()
{for(int i=0;i<16;i++) vestaBasic::auxWord[i]=
    (char)vestaDeviceType::readByte(mtEEPROM,mvarEEPPROMAddress+1+i);
```

```
return &vestaBasic::auxWord[0];}
unsigned char bOutput::getEeprom()
{return (unsigned char)vestaDeviceType::readByte
    (mtEEPROM,mvarEEPPROMAddress+17);}
bool bOutput::getInitVal()
{return (getEeprom() >> 0x0) & 1;}
bool bOutput::getInverted()
{return (getEeprom() >> 0x1) & 1;}
bool bOutput::getOnChange()
{return (getEeprom() >> 0x2) & 1;}
bool bOutput::getTimerOn()
{return (getEeprom() >> 0x3) & 1;}
bool bOutput::getTimerOff()
{return (getEeprom() >> 0x4) & 1;}
bool bOutput::getLocalTimer()
{return (getEeprom() >> 0x5) & 1;}
bool bOutput::getRemoteTimer()
{return (getEeprom() >> 0x6) & 1;}
unsigned int bOutput::getOnTime()
{return (unsigned int)(vestaDeviceType::readUint(mtEEPROM,mvarEEPPROMAddress+18));}
unsigned int bOutput::getOffTime()
{return (unsigned int)(vestaDeviceType::readUint(mtEEPROM,mvarEEPPROMAddress+20));}
unsigned char bOutput::getLocalTrigger()
{return (unsigned char)vestaDeviceType::readByte(mtEEPROM,mvarEEPPROMAddress+22);}
void bOutput::setCommonDefault(unsigned char rhs)
{vestaDeviceType::writeByte(mtEEPROM,mvarEEPPROMAddress+0,rhs);}
void bOutput::setDefEnabled(bool rhs)
{setCommonDefault(vestaBasic::setBit(getCommonDefault(),0x0,rhs));}
void bOutput::setDefService(bool rhs)
{setCommonDefault(vestaBasic::setBit(getCommonDefault(),0x1,rhs));}
void bOutput::setDefRemote(bool rhs)
{setCommonDefault(vestaBasic::setBit(getCommonDefault(),0x2,rhs));}
void bOutput::setDefInternalEvents(bool rhs)
{setCommonDefault(vestaBasic::setBit(getCommonDefault(),0x3,rhs));}
void bOutput::setDefExternalEvents(bool rhs)
{setCommonDefault(vestaBasic::setBit(getCommonDefault(),0x4,rhs));}
void bOutput::setDefManageEvents(bool rhs)
{setCommonDefault(vestaBasic::setBit(getCommonDefault(),0x5,rhs));}
void bOutput::setDefAutoLocal(bool rhs)
{setCommonDefault(vestaBasic::setBit(getCommonDefault(),0x6,rhs));}
void bOutput::setDefAutoRemote(bool rhs)
{setCommonDefault(vestaBasic::setBit(getCommonDefault(),0x7,rhs));}
void bOutput::setDeviceName(char* rhs)
{for(int i=0;i<16;i++) vestaDeviceType::writeByte(mtEEPROM,mvarEEPPROMAddress+1+(i*1),(unsigned char)(rhs[i]));}
```

El compilador crea las funciones get y set correspondientes a todos los parámetros del dispositivo, tanto los que son comunes a todos como los específicos. Desde el código de servicio se utilizan las llamadas a estas funciones encapsulando el cálculo de referencias que es el resultado del proceso mecánico del compilador.

5.3.4. Clase del servidor Web vestaHttpServer (header)

```
//Vesta compiler V1.0 28/06/2022 (c) 1989,2022
class vestaHttpServer
{
private:
//Binary resources container:
static const PROGMEM unsigned char mcolResources[2870];
static const PROGMEM char bytes_js[2799];
static const PROGMEM char ctrls_js[7514];
static const PROGMEM char err404_html[765];
static const PROGMEM char index_html[2505];
static const PROGMEM char loader_js[799];
static const PROGMEM char ls2_js[3073];
static const PROGMEM char min_css[2679];
static const PROGMEM char vesta_js[6658];
```

```
static const PROGMEM char vestaDevices_js[12823];
public:
                vestaHttpServer () ; //Constructor
void   processRequest (EthernetClient client) ; //Request from client
private:
void   processGetRequest (EthernetClient client, char *resource) ;
    //GET Request from client
void   processPostRequest (EthernetClient client, char *resource) ;
    //POST Request from client
void   answerRequest (char *rhs, EthernetClient *client) ; //Answer a GET request
void   answerPost (EthernetClient *client,char *command, unsigned int offset,
                   unsigned int length, char* data) ;
    //Answer a POST request
void   renderSystem (EthernetClient *client) ;
    //Render main system page
void   sendRO (EthernetClient *client) ;
    //Sends all Read Only info to client. Just one call.
void   sendRA (EthernetClient *client) ;
    //Sends current RA info to client. Frequent call.
void   sendEEPRO (EthernetClient *client) ;
    //Sends current EEPRO info to client. Semi-Frequent call.
void   sendHDR (EthernetClient *client) ;
    //Sends devicemap to client. Used for jScript version
void   getEEPRO (unsigned int offset, unsigned int length,char* rhs) ;
void   getRA (unsigned int offset, unsigned int length,char* rhs) ;
void   auxByteToClient (EthernetClient *client, unsigned char rhs) ;
    //Sends one byte to client in hexadecimal format
void   auxIntToClient (EthernetClient *client,unsigned int rhs) ;
    //Sends one integer to client in hexadecimal format
bool   readLine (EthernetClient *client, char *output) ;
    //Read one available line from client
bool   sendToClient (EthernetClient *client, const char *rhs) ;
    //Send PROGMEN data to http client
bool   locateRes (char *rhs, EthernetClient *client) ;
    //Searchs binary resource.
};
#endif
```

Nota: Se han omitido las constantes de compilación y la inclusión de referencias.

5.3.5. Clase del servidor Web vestaHttpServer (implementación)

```
//Vesta compiler V1.0 28/06/2022 (c) 1989,2022

//...contenido suprimido...//


//Beyond this point there is not executable code...
//==================Constants definition section==================
const unsigned char vestaHttpServer::mcolResources[]=
{
0x00,0x00,0x01,0x00,0x01,0x00,0x1A,0x1A,0x00,0x00,0x01,
0x00,0x20,0x00,0x20,0x0B,0x00,0x00,0x16,0x00,0x00,0x00,
0x28,0x00,0x00,0x00,0x1A,0x00,0x00,0x00,0x34,0x00,0x00,
0x00,0x01,0x00,0x20,0x00,0x00,0x00,0x00,0x00,0xF8,0x0A,
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x0

//...contenido suprimido...//

1,0xC0,0xF0,0x60,0x81,0xC0,0xF8,0x20,0x07,0xC0,0xF8,0x20,
0x07,0xC0,0xFC,0x00,0x0F,0xC0,0xFC,0x00,0x1F,0xC0,0xFE,0x00,
0x1F,0xC0,0xFE,0x00,0x0F,0xC0,0xFC,0x06,0x0F,0xC0,0xF8,0x1F,
```

```
0x0F,0xC0,0xF8,0x70,0x1F,0xC0,0xF0,0xE0,0x1F,0xC0,0xF0,0xF0,
0x1F,0xC0,0xF0,0xF0,0x0F,0xC0,0xF0,0x00,0x03,0xC0,0xF8,0x00,
0x03,0xC0,0xFC,0x01,0x83,0xC0,0xFF,0x0F,0xE3,0xC0,0xFF,0xFF,
0xFF,0xC0,0xFF,0xFF,0xFF,0xC0
};
const char vestaHttpServer::bytes_js[]="HTTP/1.1 200 OK\r\nServer:
Vesta V4.0\r\nContent-Type: text/javascript\r\n\r\nfunction getByte
(rhs, index) {\nvar auxValue = rhs"
".substr(index * 2, 2);\nreturn parseInt(auxValue, 16);\n}\nfunction
getBit(rhs, index, subindex) { return (getByte(rhs, index)"
" & (1 << subindex)) != 0; }\nfunction getUint(rhs, index)
{\nreturn (getByte(rhs, index)*0x100) + getByte(rhs, index+1);\n}\nf"

//...contenido suprimido...//

"rg,'Disabled',-2);\naddNavy(trg,'Service',-3);\naa=xcrt('a');
aa.textContent='Help';aa.href='#';aa.addEventListener('click',func"
"tion(){crpg=-5;window.open('https://montefaro.eu/vesta-help',
'_blank').focus();});\naa.setAttribute('typeid',-5);trg.appendChil"
"d(aa);aa.id='hlp';\n}\n";
```

Lo interesante del servidor Web es que suministra a Arduino en una estructura constante almacenada en memoria del programa el código JScript a medida de la distribucion. Una parte de este código está generada por el compilador y se recoge a continuación:

5.3.6. Archivo vestaDevices.js

El compilador genera este archivo antes de crear la clase del servidor web. El resto de los archivos permanecen invariables y simplemente se transfieren al servidor web tal cual. La razón de dividir el código en varios scripts es que alivia la carga de trabajo del módulo Arduino y permite mostrar al usuario una barra de progreso mientras carga el contenido. Esta parte del script depende de los valores calculados en la compilación y proporciona los métodos para que el programa pueda mostrar y editar la información de cada instancia de cada dispositivo.

```
function shDevice(rhs)
{
cls();
const prt = document.getElementById("maincontainer");
di=xcrt('div');di.className="row";prt.appendChild(di);
col=xcrt('div');col.className="col c3";di.appendChild(col);
col.id="chdr";hddr(col,rhs,false);
colE=xcrt('div');colE.className="col c3";colE.id="ehdr";
tit=xcrt('h3');tit.innerHTML="EEPROM";colE.appendChild(tit);
di.appendChild(colE);
colA=xcrt('div');colA.className="col c3";colA.id="ahdr";
tit=xcrt('h3');tit.innerHTML="RAM";colA.appendChild(tit);
di.appendChild(colA);
colO=xcrt('div');colO.className="col c3";colO.id="ohdr";
tit=xcrt('h3');tit.innerHTML="ROM";colO.appendChild(tit);
di.appendChild(colO);
switch(rhs.devType)
{
case 0://Controller
xParent = colO; xOffset=rhs.O;xMem='O';
adb("deviceId",0);
adb("deviceVersion",1);
adb("deviceRevision",2);
adb("slotId",3);
adi("ROMOffset",4);

//...contenido suprimido...//
```

```
adk("defEnabled",0,0);adk("defService",0,1);adk("defRemote",0,2);
adk("defInternalEvents",0,3);adk("defExternalEvents",0,4);
adk("defManageEvents",0,5);adk("defAutoLocal",0,6);
adk("defAutoRemote",0,7);
adk("not",17,0);adk("and",17,1);adk("or",17,2);adk("onRisingEdge",17,3);
adk("onFallingEdge",17,4);adk("onChange",17,5);
adb("inputs[0]",18);adb("inputs[1]",19);adb("inputs[2]",20);adb("inputs[3]",21);
xParent = colA; xOffset=rhs.A;xMem='A';
adk("remote",0,2);adk("internalEvents",0,3);adk("externalEvents",0,4);
adk("manageEvents",0,5);adk("autoLocal",0,6);adk("autoRemote",0,7);
adk("last",1,1);
break;
}
shButtons();
initRefresh();
}
function createDevice(slotId,type,ro,ee,ra)
{
switch (type)
{
case 0: //Controller
icon="https://montefaro.eu/wp-content/uploads/2021/11/Processor_16x.png";break;
case 1: //Bin Input
icon="https://montefaro.eu/wp-content/uploads/2021/11/Input_16x.png";break;
case 2: //Bin Output
icon="https://montefaro.eu/wp-content/uploads/2021/11/Output_16x.png";break;
case 3: //Meteo
icon="https://montefaro.eu/wp-content/uploads/2022/04/HeatMap_16x.png";break;
case 4: //Analogic Input
icon="https://montefaro.eu/wp-content/uploads/2021/11/Input_purple_16x.png";break;
case 5: //Analog. Output
icon="https://montefaro.eu/wp-content/uploads/2022/05/Output_16A.png";break;
case 160: //Logic Gate
icon="https://montefaro.eu/wp-content/uploads/2022/05/Connect_16x.png";break;
}
return new vestaDevice(slotId,type,ro,ee,ra,icon);
}
function hddr(trg,rhs,enable)
{
icon = xcrt('img');icon.src=rhs.icon;icon.height="24";
icon.addEventListener('click',function(){shDevice(rhs);});
trg.appendChild(icon);
colO=trg;colE=trg;colA=trg;
sp=xcrt('span');
sp.innerHTML='['+rhs.slotId+']';trg.appendChild(sp);
trg.appendChild(xcrt('br'));trg.appendChild(xcrt('br'));
switch(rhs.devType)
{
case 0://Controller
xParent = colO; xOffset=rhs.O;xMem='O';
adi("SerialId[0]",26);adi("SerialId[1]",28);
xParent = colE; xOffset=rhs.E;xMem='E';

adt("deviceName",1);


addIp("localIp",35);
addIp("mask",39);
addIp("gateway",43);
xParent = colA; xOffset=rhs.A;xMem='A';
adk("enabled",0,0);adk("service",0,1);
adk("slave",1,0);
break;
case 1://Bin Input
xParent = colO; xOffset=rhs.O;xMem='O';
adb("pin",10);
xParent = colE; xOffset=rhs.E;xMem='E';

adt("deviceName",1);

xParent = colA; xOffset=rhs.A;xMem='A';
adk("enabled",0,0);adk("service",0,1);
adk("value",1,0);
```

```
break;
case 2://Bin Output

//...contenido suprimido...//

case 3://Meteo

//...contenido suprimido...//

case 4://Analogic Input

//...contenido suprimido...//

case 5://Analog. Output

//...contenido suprimido...//

case 160://Logic Gate
xParent = colO; xOffset=rhs.O;xMem='O';
xParent = colE; xOffset=rhs.E;xMem='E';

adt("deviceName",1);

xParent = colA; xOffset=rhs.A;xMem='A';
adk("enabled",0,0);adk("service",0,1);
adk("value",1,0);
break;
}
}
function fillNav(trg)
{
addNavx(trg,'Controller',0,
'https://montefaro.eu/wp-content/uploads/2021/11/Processor_16x.png');
addNavx(trg,'Bin Input',1,
'https://montefaro.eu/wp-content/uploads/2021/11/Input_16x.png');
addNavx(trg,'Bin Output',2,
'https://montefaro.eu/wp-content/uploads/2021/11/Output_16x.png');
addNavx(trg,'Meteo',3,
'https://montefaro.eu/wp-content/uploads/2022/04/HeatMap_16x.png');
addNavx(trg,'Analogic Input',4,
'https://montefaro.eu/wp-content/uploads/2021/11/Input_purple_16x.png');
addNavx(trg,'Analog. Output',5,
'https://montefaro.eu/wp-content/uploads/2022/05/Output_16A.png');
addNavx(trg,'Logic Gate',160,
'https://montefaro.eu/wp-content/uploads/2022/05/Connect_16x.png');
addNavy(trg,'Console',-4);
addNavy(trg,'Disabled',-2);
addNavy(trg,'Service',-3);
aa=xcrt('a');aa.textContent='Help';aa.href='#';
aa.addEventListener('click',function(){crpg=-5;
window.open('https://montefaro.eu/vesta-help', '_blank').focus();});
aa.setAttribute('typeid',-5);trg.appendChild(aa);aa.id='hlp';
}
```

5.3.7. Conclusiones del compilador

Cualquier procesador moderno fabricado después del año 2020 tiene potencia sobrada para calcular en tiempo real las referencias a la memoria donde se almacenan los datos de los dispositivos. La primera versión de Vesta asumía que Arduino podría gestionar este trabajo con una pérdida de eficiencia mínima, lo que permitiría desarrollar un código modular y robusto. La experiencia ha demostrado que la única alternativa viable consiste en precalcular y definir las funciones de acceso, primero a mano, y posteriormente usando el compilador. Éste facilita enormemente la labor de desarrollo simplificando las mejoras de la estructura. Se asume por tanto que el código resultante de la compilación está pensado para *funcionar* aunque el estilo de codificación no sea el idóneo.

5.4. El compilador de Lares

Es la pieza final del proyecto. Sirve para generar el procesador domótico central, que otorga el comportamiento a la vivienda. Desde un punto de vista general los controladores Vesta son simples intermediarios entre los aparatos físicos integrados en la vivienda y una serie de objetos lógicos con entidad propia que se pueden relacionar entre sí. La abstracción de Vesta sirve también para deslocalizar todos estos dispositivos. Lo verdaderamente importante es su existencia y la función que aportan al hogar, pero no el enlace físico o el módulo Vesta con que están conectados. Si los dispositivos fuesen abonados telefónicos de la red cableada, el número de teléfono sería la imagen lógica de Vesta y el aparato el dispositivo en sí. En el momento de marcar un determinado número, el llamante ignora el camino exacto que seguirán las comunicaciones, los nodos que atravesará, la tecnología del enlace, etc. Lo importante es que cualquier abonado puede comunicar con otro abonado y Lares será el componente encargado de permitirlo.

5.4.1. Código fuente de Lares

```
using System;
using System.Threading.Tasks;
using Lorena22.VestaCompiler;
using ArduinoCompiler.Vesta.Instances;

namespace EntityCompiler
{
    class Program
    {
        static void Main(string[] args)
        {
            MainAsync(args).GetAwaiter().GetResult();
        }
        static async Task MainAsync(string[] args)
        {
#if (Execution)
            await Execute();
#else
            VersionBase auxVersion = new ArduinoCompiler.Vesta.Versions.Version2();
            await compile(true,auxVersion);
            //await compile(false);
#endif
        }

        static async Task compile(bool precompile, VersionBase vestaBase)
        {
            if (precompile)
            {
                Precompiler previo = new Precompiler();
                //await previo.addController("192.168.2.4","Coso");
                //await previo.addController("192.168.2.10", "Miele");
                //await previo.addController("192.168.2.11","Pruebas");
                //await previo.addController("192.168.2.12", "Trastero");
                //await previo.addController("192.168.2.15", "Armario");
                await previo.addController("192.168.2.200");
                previo.moduleName = "Domus";
                await previo.build("..\\..\\..\\..\\DomusOutputTemplate\\Lares",vestaBase);

            }
        }
    }
}
```

De todo el código, la parte más importante es la función **compile** en la que se creará una instancia del objeto **Precompiler** y se le añadirán las direcciones IP y un identificador opcional. Si no se especifica, el compilador tomará el asignado al módulo Vesta.

La instrucción **build** iniciará el proceso. El compilador invoca la función HDG de cada módulo Vesta obteniendo un descriptor con los dispositivos contenidos. A continuación cargará la información actual de todos esos dispositivos desde el módulo Vesta y replicará su contenido en una estructura. Esta estructura servirá para que el compilador genere una clase abstracta (virtual pura) que contendrá todos los dispositivos de todas las controladoras Vesta y una colección de referencias a cada controladora. Los nombres de los identificadores de cada dispositivo son los que el usuario ha asignado en la web de Vesta. El compilador da mensajes de advertencia si existe algún identificador duplicado en esa controladora y en cualquiera de las demás. En este caso sólo creará el primero en la salida de código.

5.4.2. Clase abstracta del controlador central

La salida de la compilación es:

```
//Autogenerated Vesta Code
using System;
using System.Collections.Generic;
using System.Text;
using Lorena22.Lares.Devices;

namespace Lorena22.Lares
{
public abstract class Domus:basePlan
{
#region Declarations
// Kernels:
internal Kernel VestaAKer;
// Devices:
internal controller VestaA {get;set;}
internal bInput DVC00popo {get;set;}
internal bInput DVC003 {get;set;}
internal bInput RedOneI {get;set;}
internal bInput BlackPulse {get;set;}
internal bInput BigWhiteI {get;set;}
internal bInput RedPulseI {get;set;}
internal bInput RedSmallI {get;set;}
internal bOutput Red2 {get;set;}
internal bOutput Red1 {get;set;}
internal bOutput White4 {get;set;}
internal bOutput White3 {get;set;}
internal bOutput White2 {get;set;}
internal bOutput White1 {get;set;}
internal bOutput BigO {get;set;}
internal bOutput Green4 {get;set;}
internal bOutput Green3 {get;set;}
internal bOutput Green2 {get;set;}
internal bOutput Green1 {get;set;}
internal bOutput Red3 {get;set;}
internal dhtVesta DVC021 {get;set;}
internal dhtVesta DVC022 {get;set;}
internal aInput VC023 {get;set;}
internal aInput VC024 {get;set;}
#endregion
// Constructor:
internal Domus()
{
initializeInstances();
mapDeviceEvents();
```

```
}
#region Initialization
protected override void initializeInstances()
{
// Initialize kernels
VestaAKer = new Kernel("192.168.2.200");
VestaAKer.devices = new ItemBase[29];
registerKernel(VestaAKer);
// Initialize devices
VestaA = new controller(VestaAKer);
DVC00popo = new bInput(VestaAKer);
DVC003 = new bInput(VestaAKer);
RedOneI = new bInput(VestaAKer);
BlackPulse = new bInput(VestaAKer);
BigWhiteI = new bInput(VestaAKer);
RedPulseI = new bInput(VestaAKer);
RedSmallI = new bInput(VestaAKer);
Red2 = new bOutput(VestaAKer);
Red1 = new bOutput(VestaAKer);
White4 = new bOutput(VestaAKer);
White3 = new bOutput(VestaAKer);
White2 = new bOutput(VestaAKer);
White1 = new bOutput(VestaAKer);
BigO = new bOutput(VestaAKer);
Green4 = new bOutput(VestaAKer);
Green3 = new bOutput(VestaAKer);
Green2 = new bOutput(VestaAKer);
Green1 = new bOutput(VestaAKer);
Red3 = new bOutput(VestaAKer);
DVC021 = new dhtVesta(VestaAKer);
DVC022 = new dhtVesta(VestaAKer);
VC023 = new aInput(VestaAKer);
VC024 = new aInput(VestaAKer);
VestaA.setReferences(0,0,0,0);
DVC00popo.setReferences(48,40,63,1);
DVC003.setReferences(70,48,99,3);
RedOneI.setReferences(81,52,117,4);
BlackPulse.setReferences(92,56,135,5);
BigWhiteI.setReferences(103,60,153,6);
RedPulseI.setReferences(114,64,171,7);
RedSmallI.setReferences(125,68,189,8);
Red2.setReferences(136,72,207,9);
Red1.setReferences(147,76,230,10);
White4.setReferences(158,80,253,11);
White3.setReferences(169,84,276,12);
White2.setReferences(180,88,299,13);
White1.setReferences(191,92,322,14);
BigO.setReferences(202,96,345,15);
Green4.setReferences(213,100,368,16);
Green3.setReferences(224,104,391,17);
Green2.setReferences(235,108,414,18);
Green1.setReferences(246,112,437,19);
Red3.setReferences(257,116,460,20);
DVC021.setReferences(268,120,483,21);
DVC022.setReferences(279,140,511,22);
VC023.setReferences(290,160,539,23);
VC024.setReferences(301,174,569,24);
}

protected override void mapDeviceEvents()
{
}
#endregion
}
}
```

La función **setReferences** sirve para localizar el contenido de cada dispositivo en el buffer de entrada al recibir datos de Vesta y para escribir estos datos en crudo en cuanto se quiera editar algún valor. Como es el módulo central domótico el que hace los cálculos de direccionamiento, el controlador Vesta queda liberado de tal función optimizando su rendimiento.

5.4.3. Un ejemplo de plan domótico

Los planes domóticos son los objetos que establecen el comportamiento de la vivienda. Deben ser una clase que herede de Lares. La clase que se muestra a continuación establece unas acciones sencillas que han servido para probar en banco el comportamiento en línea de la controladora[15].

```
using Lorena22.Lares.Devices;
namespace Lorena22.Lares
{
    internal class MainPlan:Domus
    {
        internal MainPlan():base(){}

        //Bucle principal. Aquí se coloca toda la funcionalidad que
        //deba ejecutarse cíclicamente.
        //Si devuelve False, la ejecución terminará y se creará un
        //registro de error.
        protected async override Task<bool> loop()
        {
            bool result = await base.loop();
            return result;
        }

        //Función de inicio de sistema. El procesador domótico
        //la invoca en el arranque.
        //Si devuelve False el plan domótico no se iniciará.
        protected async override Task<bool> set()
        {
            bool result = await base.set();

            return true;
        }

        //Función de respuesta al evento de un interruptor que cambia de estado
        internal async Task<bool> onSwitchChanged(ItemBase sender, byte[] buffer)
        {
            bInput boton = (bInput)sender;
            pp("Interruptor {0} {1}", boton.deviceName, buffer[0]==0?"abierto":"cerrado");
            return true;
        }

        //Función de respuesta al evento de un interruptor concreto que cambia de estado.
        internal async Task<bool> onWhiteSwitchChanged(ItemBase sender, byte[] buffer)
        {
            bool rhs = (buffer[0] != 0);
            pp("Interruptor {0} {1}", ((bInput)sender).deviceName, rhs ? "cerrado":"abierto");
            //log("Event received");
            White1.value = rhs;
            White2.value = rhs;
            White3.value = rhs;
            White4.value = rhs;
            bool result = await VestaAKer.updateRAM(0);
            //log("updateRAM");
            return result;
        }

        //Función de respuesta a una pulsación de un botón.
        internal async Task<bool> onPulser(ItemBase sender, byte[] buffer)
        {
            pp("Pulsador {0} pulsado",((bInput)sender).deviceName);
            return true;
        }

        //Función invocada en cuanto cambie la temperatura o la humedad del sensor interno
        //de una controladora Vesta.
        internal async Task<bool> onMeteoChange(ItemBase sender, byte[] buffer)
        {
```

[15]La descripción de cada bloque aparece en las líneas de comentario del código

```
            VestaAKer.RA = null;
            await VestaAKer.updateRAM(0); //Recupero información actualizada.

            pp("La temperatura es {0}º, y la humedad es del {1}%", DVC021.temp, DVC021.hum);
            return true;
        }

        //Asignación de funciones delegadas a sus correspondientes eventos
        protected override void mapDeviceEvents()
        {
            base.mapDeviceEvents();
            RedSmallI.BOnChange+=onSwitchChanged;
            Red2.BOnChange += onSwitchChanged;
            BigWhiteI.BOnChange += onWhiteSwitchChanged;
            DVC021.DHTMeasureChange += onMeteoChange;
        }


        //Funciones auxiliares para mostrar texto en la terminal del sistema o en los logs.
        internal void pp(string rhs) { Console.WriteLine(rhs); }
        internal void pp(string rhs, params object[] args) { pp(string.Format(rhs, args)); }
        internal void pp(object rhs) { pp("{0}", rhs); }
    }
}
```

Capítulo 6

Conclusiones

Los objetivos de este trabajo han sido sentar las bases de una nueva generación de domótica de altas prestaciones, acorde a los avances tecnológicos y abierta a cualquier entusiasta con nociones medias de programación. Aunque el alcance del proyecto en sí pueda parecer limitado, el camino que se abre a continuación no lo es.

6.1. Conclusiones

Se ha concebido una arquitectura domótica basada en componentes de bajo coste, aprovechando una red de transmisión eficiente cuyos componentes se pueden encontrar en cualquier tienda de electrónica o de informática y abierta a cualquier usuario. Para ello se ha desarrollado el módulo **Vesta**, que establece un puente entre el mundo físico y la abstracción lógica propia de los lenguajes de orientación a objetos.

El resultado ha sido un compilador que produce imágenes para instalar en placas Arduino Mega transformándolas en controladores domóticos (figura 6.1).

Las distintas etapas de desarrollo han condicionado la evolución desde el diseño original, más formal y fiel a las convenciones sobre el estilo de programación hasta un código más centrado en la eficiencia, en el que predominan los valores precalculados y las estructuras ya desarrolladas para ahorrar tiempo de procesador. Finalmente se decidió volver a la claridad del código estructurado y bien documentado creando una herramienta que permite generar el código optimizado para el microcontrolador como código objeto o de transición y otra que es capaz de sondear la información almacenada en varios controladores **Vesta** para construir un sistema central a medida a partir del cual se establece el comportamiento de la vivienda inteligente, pero desde una perspectiva lógica, tratándola como si de una colección de objetos lógicos se tratara. En la figura 6.1 se muestra la primera controladora Vesta A Plus que entró en producción. Está instalada en el armario eléctrico principal y lleva en funcionamiento ininterrumpido desde comienzos de 2021. A la finalización de este trabajo será actualizada a la nueva versión e incorporada a Lares.

6.2. Impacto social y medioambiental

Este producto aspira a democratizar el acceso a la automatización inteligente de los hogares de la misma forma que internet lo hizo con el mundo de las comunicaciones, dando los mismos privilegios de acceso a los particulares que a las grandes corporaciones. La instalación de controladores Vesta y procesadores centrales Lares en hogares de clases medias puede suponer una

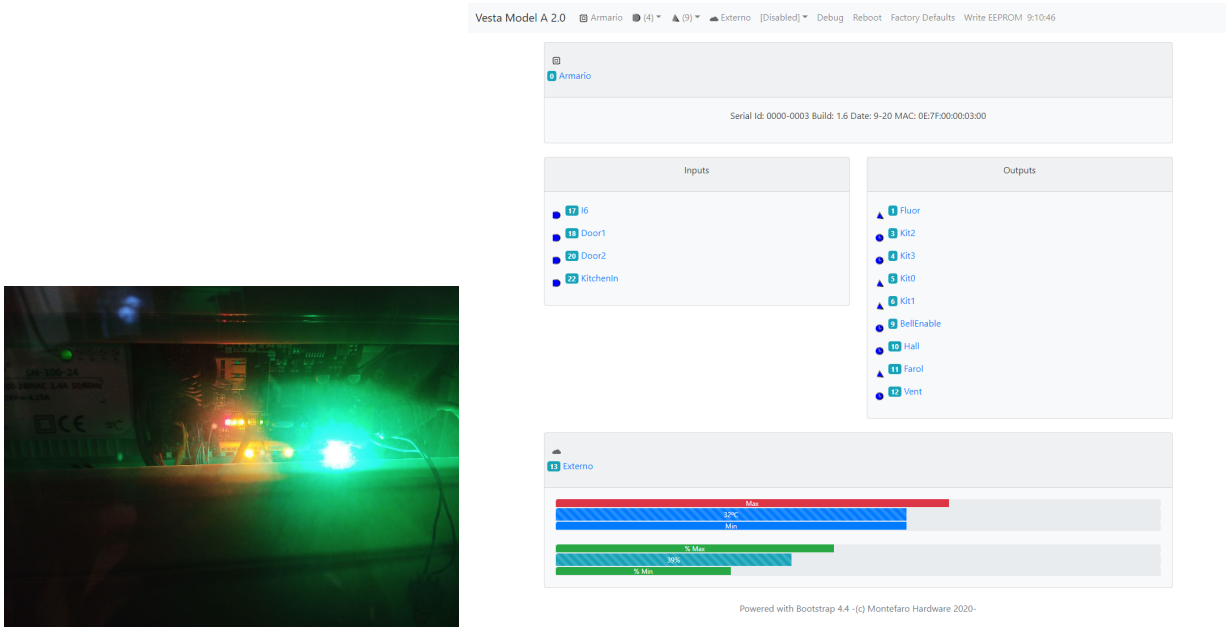


Figura 6.1: Controladora Vesta A Plus del salón y cocina.

oportunidad de especialización para oficios técnicos como el de electricista, convirtiéndolos en asesores domóticos en potencia, capaces de proyectar una reconversión de un domicilio existente o de prestar servicio técnico tras un breve periodo de formación.

Surge también un nuevo campo abierto para la comunidad de desarrolladores que a partir de las clases abstractas de **Lares** pueden construir librerías de código con comportamientos domóticos centrados en la eficiencia energética, la seguridad, el confort doméstico y el intercambio de datos con el entorno para mejorar las prestaciones de los hogares. La sana competencia entre tecnologías de software aplicadas a la domótica y basadas en la misma plataforma dará lugar a la creación de nuevas empresas cuyo productos de software serán tan útiles o más en el hogar como lo puede ser un nuevo mueble para el salón o una alfombra para el dormitorio.

Los grandes negocios del hogar podrán incorporar a su catálogo los servicios domóticos por licencia de un solo uso o por suscripción, integrando los dispositivos ya existentes en el hogar con estas librerías o poniendo en el mercado nuevos dispositivos, de la misma forma que diferentes distribuidores ponen a la venta enchufes, interruptores de la luz o bombillas, todos ellos compatibles entre sí y libres de ataduras a una marca o a un determinado sistema del que no se puede salir.

La centralización de todas las formas de domótica en un mismo sistema a través de paquetes de software evitará tener que adquirir los mismos aparatos varias veces para una alarma, por tener un mando a distancia o por razones de seguridad. A través de Lares y de Vesta, los mismos aparatos adquiridos a distintos fabricantes, serán empleados por todos los paquetes de software que se contraten.

Finalmente, la aplicación de la inteligencia artificial y, especialmente los algoritmos de aprendizaje automático, sentarán las bases de un nuevo tipo de hogar inteligente diseñado para aprender de las necesidades de sus propios habitantes, anticipándose a sus deseos e involucrando absolutamente a todos los dispositivos eléctricos y electrónicos a su disposición para elevar con ellos la eficiencia y el confort del hogar.

La mejor vivienda inteligente no es la que tiene muchos botones de colores o persianas que se abren solas sino aquella en la que la existencia es más sencilla, agradable, segura y económica, sin llegar a saber exactamente por qué.

6.3. Lineas futuras

Este trabajo constituye un pequeño hito en un camino mucho más grande. A partir de este punto comienzan las bifurcaciones, que serían objeto de otro trabajo de fin de grado, esta vez de ciencias económicas o empresariales pues será necesario tomar algunas decisiones que condicionarán el futuro. ¿Se debe comercializar? ¿Es sostenible crear una empresa que viva solamente con donativos de la comunidad? ¿Debe ser un proyecto totalmente libre?

Cuando se construyó el compilador se generó sin pretenderlo un mecanismo anti-copia. Cada controladora Vesta tiene un número de serie único, que sirve para calcular la dirección MAC[1]. Esto quiere decir que si se clona el contenido de un controlador Vesta, la nueva tarjeta clonada no funcionará en la misma red ya que ambas tendrán la misma dirección MAC.

Una fórmula es crear las imágenes a través de un servicio web en tiempo real de forma automatizada a cambio de crear una cuenta y que Lares valide el acceso para continuar con el servicio domótico. Las posibilidades de monetización y control son muchas.

Otra opción consiste en coordinar el proyecto sin ánimo de lucro y crear una empresa proveedora de software para Lares en competencia con las demás. En todos los casos habría que estudiar la demanda potencial, la penetración en el mercado y las acciones de marketing o publicidad que deberían realizarse.

Se escoja el camino que se escoja, los dos objetivos principales de este trabajo, que se consideran superados son la culminación de los requisitos para obtener el Grado en Ingeniería Informática en la Escuela Técnica Superior de Ingeniería Informática y el reseñado en la siguiente sección.

6.4. Una perspectiva egoísta

De todos los clientes potenciales a los que el ecosistema *Lares/Vesta* puede simplificar la vida, el más exigente de todos es también el autor de este trabajo. Ya desde el año 2004, en que se instaló el primer computador con aspiraciones a automatizar las funciones del hogar, no ha cesado la investigación en el laboratorio más exigente de todos, que es un hogar real en funcionamiento. La vida del día a día acaba revelando qué prestaciones domóticas suponen una mejora y cuáles, pese a su aparente espectacularidad, quedan relegadas al olvido por la falta de utilidad. En una vivienda que sirve de hogar para una familia es donde se aprecian las consecuencias de un error de programación, de un mal diseño eléctrico, de una interferencia entre aparatos que gestionan

[1]En las redes de computadoras, la dirección MAC (Media Access Control) es un identificador de 48 bits que corresponde de forma única a una tarjeta o dispositivo de red. Se conoce también como dirección física y es única para cada dispositivo. Más información en https://help.gnome.org/users/gnome-help/stable/net-macaddress.html.es

el mismo recurso, de ese tipo de sucesos que hacen que un algoritmo aparentemente correcto encuentre su caso inesperado...

Que el hogar sea más cómodo se convierte en la única meta y cuando cultivando una simple afición de tiempo libre se tiene la oportunidad de probar nuevos aparatos y ponerlos a funcionar en la casa, es cuando se descubren nuevas utilidades de componentes que aparentemente servían para otra cosa. Estos estudios han ampliado considerablemente las posibilidades de lo que hace unos años se podía imaginar. Asignaturas como Sistemas Operativos, Aprendizaje Automático, Ampliación de Sistemas Inteligentes, Teoría de los Lenguajes de Programación, Seguridad Informática, Procesadores del Lenguaje, las tres de Ingeniería de los Computadores, Diseño y Administración de Sistemas Operativos e incluso Estadística, han ayudado a mantener una visión de conjunto mucho más sólida de cómo se podían alcanzar objetivos que antes sólo se podían imaginar. Y no por el contenido de las asignaturas en sí, sino porque los conceptos que engloban son versátiles... La clase vestaKernel es más que sospechosamente parecida en su funcionamiento al kernel de Linux y la forma en que se crean y consultan las tablas de dispositivos de la controladora Vesta son idénticas que las que los sistemas operativos UNIX tienen para gestionar los procesos... Las diferencias conceptuales son grandes, las lógicas mínimas.

Al final, independientemente de los nebulosos beneficios sociales y económicos que pueda deparar esta *carrera*, la gran aportación, el gran regalo ha sido conseguir la varita mágica para transformar todo lo que este cliente exigente era capaz soñar en componentes de un producto estable, potente y versátil, que incluso podría acabar convertido en un negocio.

Pero el objetivo más ambicioso de todos es que el hogar de este cliente egoísta se actualice con esta domótica. El premio más anhelado, la guinda del pastel, será poder emplear el tiempo que hasta ahora se llevaban las asignaturas y las prácticas en instalar, configurar y desarrollar código para las 12 nuevas controladoras Vesta que están proyectadas para actualizar esta vivienda. Este hogar en permanente evolución por fin, y gracias a la Universidad, se va a convertir en el escaparate de un producto maduro del que a partir del final del verano del año 2022, sólo se desarrollará código para crear la inteligencia que los exigentes habitantes de este inmueble demandan.

Y tras haber descansado un poco de estudiar, lo más probable es que se retome la actividad cursando el máster, porque ya antes de haber hecho la primera matrícula, está muy claro que el trabajo del final de los estudios tiene que ser obligatoriamente una librería de aprendizaje automático para Lares. Y en el primer punto de la bibliografía vendrá una referencia a este proyecto.

Bibliografía

[1] C. Tabernier, *Montajes Domóticos*. S.A. Ediciones Paraninfo, 1995.

[2] J. M. Angulo Usategui, *Electrónica Digital Moderna*. S.A. Ediciones Paraninfo, 1984.

[3] J. Sánchez, *Microcontroller Programming*. CRC Press, 2006.

[4] D. Borrajo Millan, *Aprendizaje Automático*. Sanz y Torres S.L., 2008.

[5] J. T. Palma Méndez, *Inteligencia Artificial: Técnicas Métodos y Aplicaciones*. MC-Graw Hill, 2008.

[6] R. Sethi, *Compiladores: Principios, técnicas y herramientas*. Pearson Addison-Wesley, 2008.

[7] J. M. Díaz Martínez, *Fundamentos básicos de los sistemas operativos*. Sanz y Torres S.L., 2011.

[8] R. S. Pressman, *Ingeniería del software*. McGraw Hill, 2021.

[9] V. M. Dominguez Rivas, *Plantilla TFG ETSISI UPM*. ETSISI, 2020.

Apéndice A

Detalles del diseño

A.1. Microcontrolador Vesta

Es un aparato que sirve para relacionar los elementos eléctricos de la casa con el software de control domótico. A nivel general se compone de un procesador, circuitería de adaptación de niveles eléctricos, módulo de comunicaciones y firmware, que es el objetivo de este proyecto.

A.1.1. Diseño

El controlador se ha diseñado pensando, fundamentalmente, en la versatilidad. El usuario debe poder configurar el comportamiento...

A.1.2. Pasos previos

En versiones anteriores se diseñó este componente pensando en la versatilidad. El objetivo era que el usuario pudiera configurar el comportamiento de los dispositivos bajo control de una forma intuitiva. El análisis previo establecía que el firmware respondiera a las peticiones del usuario y del sistema domótico calculando los valores de los parámetros de cada dispositivo en tiempo real.

Al liberar el primer prototipo se constató que el procesador ARM de 16 bits de Arduino era demasiado lento; tanto que al responder a una simple petición de información por medio del protocolo HTTP, el módulo quedaba congelado y no era capaz de responder ni siquiera a las peticiones de más prioridad. Entonces se hicieron pruebas con un esquema de interrupciones asumiendo la lentitud de la Web como un *mal necesario* resultando un nuevo fracaso que afectaba gravemente a la escalabilidad de la arquitectura. Desde el primer momento el controlador debía ser robusto ágil y fiable y el resultado se alejaba considerablemente de las expectativas.

Se reformuló la estrategia dando máxima prioridad a la eficiencia por encima de la legibilidad del código e incluso asumiendo que el listado resultante ni siquiera tenía por qué ser legible y mantenible para un humano. El autor recordaba el compilador CCM *Concurrent Modula 2*, que alcanza multitarea a nivel de software mediante emulación incluso en computadores con procesadores que no tienen estas capacidades. Para ello, CCM compila el código generando un programa en Modula-2 estándar y no concurrente, que al ejecutarse, se comporta como se espera. El código Modula-2 producido se ignora y como es generado automáticamente no admite mantenimiento ni depuración porque no es más que un paso intermedio entre el auténtico programa CCM y el ejecutable final.

A.1.3. El compilador Vesta

Tras unas pruebas preliminares de velocidad se pudo comprobar que la plataforma Arduino podía ser tan potente como se esperaba si todos los valores necesarios ya estaban precalculados y si se generaba un código C++ *tosco* aunque efectivo. Entonces se abandonó la idea de crear un programa universal y se centraron los esfuerzos en generar las imágenes de código fuente C++ de cada una de las unidades controladoras a partir de un compilador intermedio que se convirtió en el auténtico objetivo del trabajo.

El*código fuente* son dos clases C# denominadas **versión** y **distribución**. La primera enumera todos los posibles tipos de dispositivos que puede albergar el hardware y la segunda contiene una lista de los que están contenidos en esa unidad concreta, en la que se pueden definir propiedades por defecto para cada uno.

A.1.4. Las tres memorias

El compilador asigna memoria en el circuito microcontrolador para cada dispositivo. Esta memoria es de tres tipos:

1. **ROM**

   Que contiene los datos propios de cada instancia del dispositivo y que no se pueden modificar durante toda la vida del controlador. Por ejemplo, los pines del hardware o los valores iniciales por defecto de algunos datos.

2. **RAM**

   Son datos contenidos en la memoria de acceso aleatorio que podrán ser modificados y consultados asumiendo que se perderá la información al restablecer la alimentación. El valor actual de una salida digital, un contador o un totalizador son ejemplos de parámetros que deben ser almacenados en este tipo de memoria.

3. **EEPROM**

   Es la información de configuración. Son datos similares a los de la memoria ROM, pero que el usuario puede cambiar en algún momento y quedan con el nuevo valor incluso al restablecer la alimentación. El nombre de cada dispositivo, los flags de los diferentes modos de funcionamiento o un valor de calibración son ejemplos de valores para EEPROM

Todos los dispositivos del mismo tipo tienen los mismos parámetros en las tres áreas de almacenamiento y cada instancia se diferencia de las demás por tres parámetros almacenados en ROM que contienen las direcciones de comienzo de cada tipo de memoria. Es labor del compilador hacer los cálculos de offset de cada parámetro y generar el código correspondiente en una instancia de una clase llamada **vestaConfig**.

Para ahorrar la escasa memoria RAM disponible para datos en Arduino, toda la información no modificable como cadenas de caracteres, constantes o los valores contenidos en el área ROM se

decora con un parámetro especial denominado **PROGMEM** que fuerza al compilador de Arduino a colocar estos contenidos en la misma memoria flash que alberga el código del programa, mucho más abundante que RAM y EEPROM juntas.

El segundo prototipo y primer desarrollo generado con el nuevo compilador se mostraba mucho más rápido que todo lo anterior, aunque el microcontrolador daba síntomas de corrupción de memoria al asignar una determinada cantidad de dispositivos. Tras un arduo trabajo de depuración se comprobó que cada instancia de cada dispositivo ocupaba la memoria dos veces, una para el propio objeto (Arduino no tiene el Heap de C++ para los objetos que se crean dinámicamente), y otra para los datos que éste contenía.

Entonces se rediseñó el compilador convirtiendo los objetos de dispositivo originales (*vestaItem*) en objetos que asumían la funcionalidad de todas las instancias del mismo tipo de dispositivo (*vestaTypeDescriptor*). Con esta nueva forma de proceder, cada invocación de un dispositivo se realiza llamando a la función **getDevice(unsigned char slotId)** que devuelve un puntero al tipo del dispositivo y antes asigna los valores de offset de las tres memorias a la instancia requerida. Estos valores están almacenados en un array constante que también forma parte de **vestaConfig**.

A.1.5. Servidor HTTP

La separación entre el tipo del dispositivo y el código que le proporciona su funcionalidad también permitió incrementar considerablemente la eficacia del servidor HTTP incorporado. A partir de los datos almacenados en **vestaConfig**, el servidor Web traslada todo el proceso al frontend dejando en manos del navegador la lectura, asignación y modificación de los valores de las tres memorias. Si bien no ha sido posible incrementar la velocidad de carga de la página, sí es posible hacerla rápida para gestionar todos los parámetros de configuración e incluso acceder al terminal de depuración dando un control eficiente y robusto del microcontrolador. El programa en JScript, contenido y transferido al navegador cuando la página carga por primera vez, hace una llamada a la función HDR que responde con la estructura de la controladora codificada en ASCII. Esta estructura es la que informa al código JScript del número y tipo de dispositivos instalados. Una vez adquirida la información, el programa principal alterna las llamadas a las funciones ROM, RAM y EEPROM que devuelven respectivamente los contenidos de las tres memorias, rellenando el área de presentación con la información en tiempo real.

Dependiendo del tipo de memoria el código JScript permite también alterar los valores actuales haciendo el correspondiente cambio en cada control para la memoria RAM o editando estos datos y pulsando el botón *submit* para los datos en EEPROM.

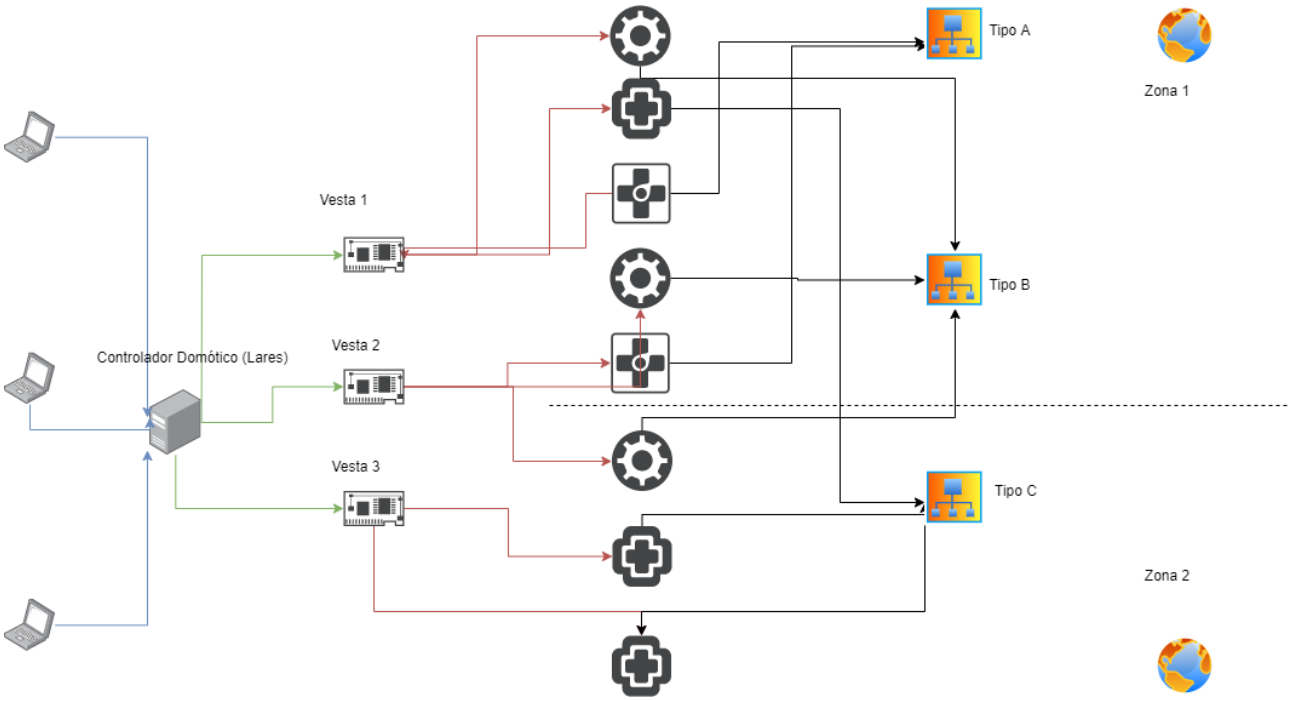
Dado lo ligero de los paquetes de datos intercambiados, el módulo Vesta puede atender a las peticiones HTTP y a sus correspondientes funciones sin interferencia aparente, e incluso servir información actualizada a varios equipos al mismo tiempo.

Apéndice B

La abstracción de Lares

Lares es una abstracción para gestionar un conjunto de dispositivos físicos conectados a varios módulos controladores programables según el sistema **Arduino**. Cada módulo Vesta es capaz de controlar uno o más dispositivos físicos que funcionan de forma independiente unos de otros.

B.1. Estructura



La relación entre los componentes se realiza a tres niveles: Físico, lógico y zonal.

B.1.1. Nivel Físico

En verde para la red y rojo para las conexiones, se compone de un sistema general de control denominado **Lares** o *Controlador Domótico* y varias tarjetas controladoras **Vesta**, conectadas físicamente a los dispositivos bajo control.

B.1.2. Nivel lógico

El tipo de los dispositivos les otorga las propiedades, métodos y eventos. Desde este punto de vista, cada dispositivo físico está relacionado con un objeto que es instancia de ese tipo. Estos objetos tienen sus respectivas listas de propiedades, sus métodos y también los eventos que pueden generar según sus características.

En la imagen se definen tres tipos de objetos y las relaciones de herencia van más allá de la controladora **Vesta** a la que están conectados físicamente. Desde el punto de vista del controlador domótico es irrelevante cómo se realizaron esas conexiones.

B.1.3. Nivel Zonal

Es visible sólo desde el controlador domótico. Todos los dispositivos lógicos tienen una propiedad que les permite pertenecer a un grupo zonal o zona. Una zona es un tipo particular de dispositivo lógico que contiene varios dispositivos que se encuentran en una misma ubicación. El nivel zonal es jerárquico: Una zona puede contener a otras zonas.

Por medio del nivel zonal es posible definir relaciones entre dispositivos pertenecientes a una misma habitación o edificio agregando una capa de pertenencia para las iteraciones. Por ejemplo, es posible enviar un comando a *todas las luces de una sala* sin especificar cuántas son ni cuáles son las controladoras físicas a las que están conectadas. Las iteraciones zonales seguiran funcionando aunque se agreguen nuevos dispositivos o se eliminen parte de los que están.

B.1.4. Responsabilidad de los aparatos de control

1. Lares (Controlador domótico): Nivel zonal y nivel lógico.
2. Vesta: Nivel físico (y lógico sólo en modo local).

El modo local es la capacidad que tienen las controladoras Vesta de ejecutar programas sencillos como parte de dispositivos complejos, que gestionan a otros dispositivos simples y en dispositivos *a medida* hechos por el propio desarrollador.

B.2. Definiciones

B.2.1. Controlador domótico

Es un software que se ejecuta en una máquina independiente conectada a la misma red local en la que están todos los dispositivos **vesta**. La funcionalidad de este componente queda fuera del alcance de este proyecto más allá de la librería **Vesta**.

B.2.2. Librería Vesta

Se llama así la capa de abstracción que permite al **controlador domótico** gestionar los dispositivos de la red domótica como objetos. Estos objetos se clasifican por familias de acuerdo a los distintos tipos de dispositivos lógicos existentes que exhiben unas propiedades con las que se puede obtener toda la funcionalidad de los dispositivos físicos al otro lado de la abstracción. Además, los objetos proporcionan métodos y ofrecen algunas propiedades de escritura con las que el controlador domótico puede realizar cambios en su estado.

B.2.3. Dispositivo físico

Es cualquier elemento del hogar que funciona con corriente eléctrica y actúa bajo el control del sistema domótico. Son dispositivos físicos los interruptores de la luz, los aparatos de calefacción, cualquier lámpara de la casa, los pulsadores del timbre, etc.

Los dispositivos físicos se pueden clasificar en función del flujo de la información como **de entrada**, si captan estados del mundo real, **de salida**, cuando cambian el entorno del hogar, o **mixtos**, cuando pueden actuar como de entrada o de salida indistintamente.

Cada dispositivo físico puede adoptar una tecnología, voltaje o latencia de operación diferente en función de su constitución, siendo misión del circuito electrónico de la controladora comunicarse o alimentarlo de forma transparente al sistema.

B.2.4. Dispositivo lógico

Se denomina así a la **abstracción** que contempla la operativa de una familia entera de dispositivos físicos similares. Por ejemplo todas las luces pueden encenderse o apagarse, independientemente de la tecnología (led, incandescente, halógena,...) o el voltaje que requieran. En la abstracción lógica, una lámpara es un dispositivo de *salida binaria* que puede tener dos estados (apagado y encendido).

El objetivo de la tecnología **Vesta** es gestionar el funcionamiento de todos los dispositivos físicos a través de estas abstracciones, ocultando los detalles operativos del nivel físico al usuario humano.

Cada familia de dispositivos se identifica por un número entre 0 y 255 (0x00 - 0xFF), de acuerdo a la siguiente tabla:

Id	constante	tipo	notas
0x00	VDTController	Controladora Vesta	La propia controladora
0x01	VDTInput	Entrada binaria	Ej. Un pulsador o interruptor
0x02	VDTOutput	Salida binaria	Ej. lámparas, timbres, etc.
0x03	VDTDHT	Meteo básico	Sensor temperatura y humedad
0x04	VDTGate	Persiana	Motorizados que abren y cierran
0x05	VDTRfid	Sensor RFID	Lector de tarjetas contactless
0x06	VDTAnalogicInput	Entrada analógica	Sensores de todo tipo
0x07	VDTAnalogicOutput	Salida analógica	Luces atenuables, motores, etc.
..			
..			
0x80	VDTParking	Gestor de estacionamiento	Control básico de un parking.
0x81	VDTElev	Controlador de ascensor	Control básico de un ascensor.
0x82	VDTAlarm	Alarma antirrobo	Módulo básico de alarma.
..			
..			
0xFE	VDTCustom	Desarrollo propio	control no estándar.
0xFE	VDTTemplate	Plantilla	Nuevos dispositivos.
0xFF	VDTNull	Dispositivo Nulo	Dispositivo Nulo.

Las familias de dispositivos lógicos se clasifican en tres grupos:

Básicos

Con Id entre 0x01 y 0x7F (126 posiciones). Se trata de dispositivos elementales de uso sencillo para el control de todo tipo de elementos básicos de una casa, desde lámparas a aparatos de aire acondicionado.

Complejos

Son módulos compuestos de dispositivos básicos y que controlan sistemas más elaborados o instalaciones que se pueden gestionar de forma modular. Sirvan como ejemplo los dos que están definidos en esta versión y que son un aparcamiento que comprende una o varias puertas motorizadas (VDTGate), pulsadores para encender las luces (VDTInput), luminarias (VDTOutput), sensores varios (VDTAnalogicInput) y lectores contactless (CDTRfid) en un solo módulo.

Que estas instalaciones estén definidas como dispositivo lógico significa que un solo módu-

lo **Vesta** puede gestionarlas de forma local sin necesidad de formar parte de una estructura domótica superior, aunque todos sus componentes pueden ser operados por separado si se requiere, asumiendo un control total desde la parte domótica.

Especiales

Son cuatro tipos: VDTController, VDTCustom, VDTTemplate y VDTNull.

1. VDTController: Es la propia controladora, que actúa como un dispositivo más.
2. VDTCustom: El tipo 0xFE es el único para el que el sistema domótico no tiene un dispositivo lógico predefinido. Es el usuario avanzado quien implementará toda la operativa de este tipo de dispositivo suponiendo que no exista ninguna familia lógica asimilable al desarrollo que se pretende controlar.
3. VDTTemplate: La librería de Vesta incluye un tipo especial de dispositivo para generar nuevos tipos.
4. VDTNull: El controlador ignorará este tipo al iniciar el sistema de forma que este tipo de dispositivo no aparecerá en la configuración ni en el sistema domótico.

B.2.5. Slot

Cada controladora vesta puede contener uno o más dispositivos lógicos. El **slot** es el índice de cada uno de ellos y se representa con un número natural entre corchetes [x] como el índice de un vector. El tipo de dispositivo lógico que ocupa cada slot queda definido de fábrica en la controladora y no se puede modificar desde el software ni en la configuración web.

B.2.6. Distribución

Es un modelo de controladora Vesta. A nivel físico diferentes modelos tienen aspectos distintos, pueden funcionar con voltajes diferentes y han sido desarrollado para propósitos también distintos.

A nivel lógico, una distribución se diferencia de otra por la asignación de sus slots.

Por ejemplo, la distribución denominada **ModelA** tiene los siguientes componentes:

Existen dos tipos de distribuciones. Las **canónicas** son las de propósito general, están documentadas y adaptadas para su uso por el gran público y las **custom** que son desarrollos particulares o comerciales enfocadas en un tipo específico de función.

Los entornos domóticos compatibles con **Vesta** contienen la documentación necesaria para configurar las distribuciones canónicas en modo *plug and play* y las custom requieren que los integradores suministren al sistema una lista con la asignación de dispositivos lógicos a los slots.

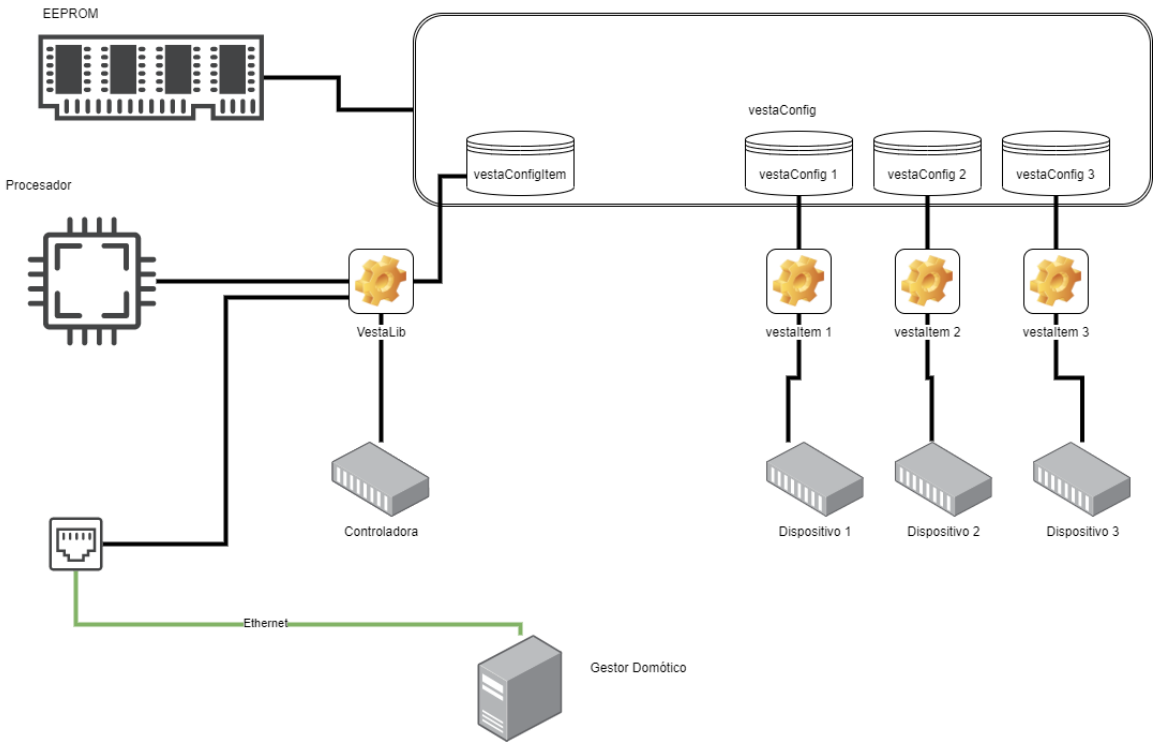


Figura B.1: Estructura del hardware

slot	tipo	nombre
0	VDTController	-
1	VDTDHT	Internal
2	VDTDHT	External
3	VDTInput	in0
4	VDTInput	in1
5	VDTInput	in2
6	VDTInput	in3
7	VDTInput	in4
8	VDTInput	in5
9	VDTInput	in6
10	VDTInput	in7
11	VDTOputput	rele0
12	VDTOputput	rele1
13	VDTOputput	rele2
14	VDTOputput	rele3

Tabla B.1: Ejemplo de distribución canónica Vesta

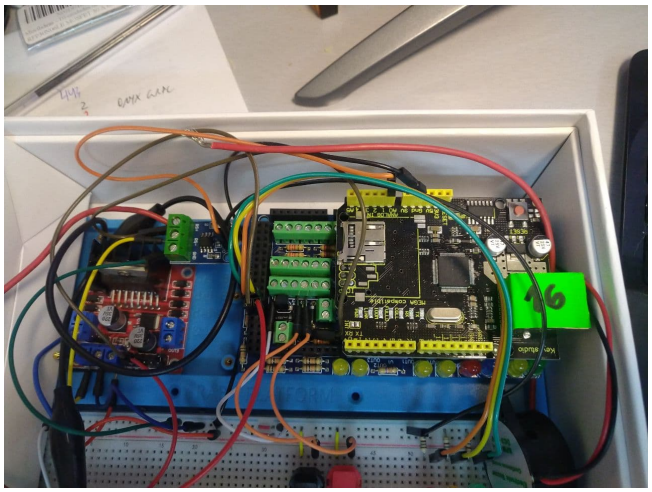


Figura B.2: Primeras pruebas de conexión con el prototipo de Vesta A

Una misma distribución física se puede utilizar para albergar varias distribuciones lógicas posibles. Ya que, por ejemplo, la distribucion *ModelA* dispone de ocho entradas y cuatro salidas, sería posible implementar con la misma PCB un solo dispositivo VDTGate.

En el momento de la redacción existen cuatro distribuciones canónicas:

1. Tipo A: dos sensores de clima, ocho entradas binarias y cuatro salidas binarias.
2. Tipo B: un sensor de clima, 8 entradas binarias y 16 salidas binarias.
3. Tipo AH: dos sensores de clima, 11 entradas binarias (3 pulsadores en la propia placa, 8 en los puertos) y 12 salidas binarias.
4. Tipo A Gate: igual que el tipo A, implementando también un módulo de persiana.

Todas estas distribuciones, excepto el Tipo B, tienen todos los componentes necesarios para incorporar también un dispositivo RFID.

B.3. Dispositivo Lógico

Son los elementos bajo control en el sistema domótico, accesibles desde el controlador general o controlador domótico.

Apéndice C

Dispositivos

Un edificio inteligente contiene aparatos mecánicos, eléctricos o electrónicos que actúan como los sentidos o los músculos en un organismo vivo. La filosofía del controlador **Vesta** consiste en proporcionar una abstracción lógica por cada uno de los tipos de estos dispositivos físicos.

C.1. Dispositivo genérico

Todos los dispositivos heredan o implementan las propiedades y métodos del dispositivo genérico aunque no existe un dispositivo genérico como tal.

C.1.1. Propiedades de Solo Lectura

Están definidas en la distribución y no se pueden modificar.

deviceId (unsigned char, pos 0)

Es el número del tipo de dispositivo (0 para la controladora, 1 para la entrada binaria, ...) Cada familia de dispositivos lógicos tiene un valor diferente.

deviceSlot (unsigned char, pos 1)

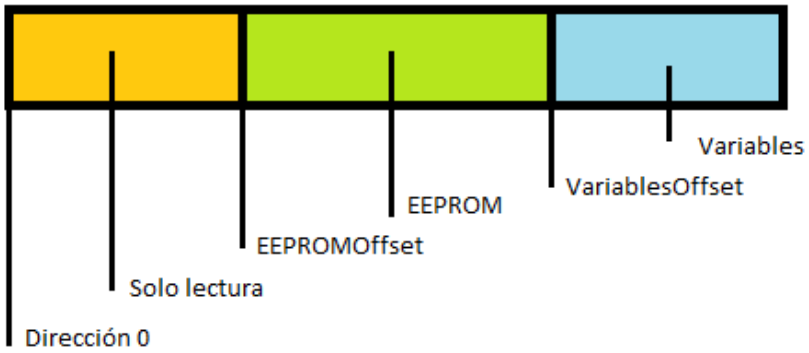
Es el número de slot que ocupa este dispositivo en la controladora. No puede haber dos dispositivos con el mismo número de slot en la misma controladora y no se pueden dejar números sin utilizar. Una controladora que aloje 10 dispositivos tendrá slots numerados del 1 al 10, siendo el 0 el de la propia controladora.

deviceVersion (unsigned char, pos 2)

Versión del tipo de dispositivo. Se contempla la posibilidad de dotar a los distintos dispositivos de nuevas funcionalidades que pueden afectar a las posiciones de memoria de los datos que almacenan, a las propiedades que tienen o a la forma de responder ante los comandos. Mediante este valor y el siguiente, la librería Vesta que maneja el controlador domótico podrá comunicarse con el dispositivo siguiendo el protocolo correcto.

deviceRevision (unsigned char, pos 3)

El número de versión se compone con el parámetro anterior y con éste.



EEPROMOffset (unsigned char, pos 4)

Toda la información asignada a este dispositivo reside en un vector de bytes en memoria RAM. En ella están los datos de solo lectura, a continuación los de la EEPROM y después los de las variables.

VariablesOffset (unsigned char, pos 5)

Dirección del vector donde residen los datos variables, que no se conservarán en un reinicio o en un fallo de alimentación.

memoryTotalSize (unsigned char, pos 6)

Tamaño del vector de datos. Es igual a la última dirección del vector de datos más uno. Sirve para calcular el tamaño de los buffers en las operaciones de transferencia de los datos del dispositivo.

resto de valores

A continuación cada familia de dispositivos lógicos tendrá valores de solo lectura propios de sí misma.

C.1.2. Propiedades EEPROM

Son valores de configuración propios de cada uno de los dispositivos lógicos que se pueden modificar desde la librería Vesta, desde la web de configuración o desde la consola del puerto

serie y que no se pierden en caso de fallo de alimentación o reinicio de la controladora.

Importante: Las direcciones en memoria se calculan sumando el valor de la posición del parámetro al valor EEPROMOffset.

enabled (pos 0 bit 0)

Habilitación del dispositivo. Cuando está a *True* funcionará normalmente y cuando está a *false* quedará deshabilitado.

service (pos 0 bit 1)

Modo de servicio. Sólo es relevante para los dispositivos con funcionamiento automático. En modo de servicio se desactivará el automatismo aunque se podrán seguir utilizando de forma manual.

remote (pos 0 bit 2)

Modo remoto. Un dispositivo que tenga el parámetro remote a *false* sólo procesará las órdenes emitidas desde otros dispositivos del mismo controlador o desde la web de configuración, pero no obedecerá los comandos que lleguen desde el controlador domótico.
Este modo es especialmente necesario para dispositivos complejos que manejan otros dispositivos más simples en la misma controladora. El modo remote impide que el controlador domótico pueda alterar el valor de ciertas partes sensibles bajo control del dispositivo complejo, como por ejemplo en el caso de los motores que abren o cierran una persiana; si el controlador domótico pudiese ejecutar un comando de conexión ignorando el estado de los sensores que indican si la puerta está ya abierta o cerrada se podría producir una avería.

internalEvents (pos 0 bit 3)

El dispositivo lógico puede emitir eventos internos. Un evento interno es un mensaje que se envía cada vez que ocurre un suceso al resto de dispositivos en la misma controladora. Suele utilizarse para que un dispositivo simple comunique el suceso al dispositivo complejo del que forma parte.

enternalEvents (pos 0 bit 4)

El dispositivo lógico puede emitir eventos externos a través de mensajes UDP por el puerto Ethernet. Suele servir para informar al controlador domótico o a otras controladoras Vesta de la ocurrencia de un suceso. Ya que cada uno de estos sucesos hará que se genere un mensaje en la red, se recomienda deshabilitar este flag para todos los dispositivos cuyos cambios de

estado no afecten al sistema domótico, como por ejemplo los propios de un dispositivo simple que trabaja para otro dispositivo complejo en la misma controladora.

Un mismo dispositivo lógico puede tener habilitados los eventos internos, los externos, los dos o ninguno.

manageEvents (pos 0 bit 5)

El dispositivo lógico puede escuchar los eventos que se producen desde otros dispositivos o desde la red. Es un flag pensado en la optimización de código. En el momento en que se recibe un evento, el controlador realiza una iteración por todos los dispositivos instalados comprobando este valor y, en caso de estar habilitado, ejecuta el código de evento. Ajustarlo a *false* evitará ejecutar innecesariamente una parte del programa y perder el tiempo de proceso.

autoLocal (pos 0 bit 6)

Transcurrido un tiempo (ajustable) sin recibir eventos desde ethernet, el dispositivo pasará automáticamente a modo local (flag remote = *false*). Es útil para mantener el funcionamiento domótico en un modo degradado en caso de fallo en el controlador domótico. Por ejemplo, si en una habitación se configura un dispositivo complejo de encendido y apagado de luces con las pulsaciones del interruptor, puede interesar que sea el controlador domótico el que se haga cargo del automatismo, pero si falla o se corta la red, pasado un tiempo, la luz seguirá funcionando igualmente de forma local.

Una vez que el dispositivo pasa al modo local, sólo podrá pasar a modo automático desde la web de configuración, mediante la consola serie o por una orden explícita del sistema domótico.

autoRemote (pos 0 bit 7)

Funcionando en modo local (flag remote = *false*), el dispositivo pasará automáticamente a modo remoto en cuanto el controlador reciba el primer mensaje del sistema domótico aunque no sea un comando dirigido explícitamente a él. Se utiliza para restaurar automáticamente el modo remoto en cuanto se reestablece el funcionamiento de la red o del controlador domótico.

deviceName (char[16], pos 1)

Nombre del dispositivo. Este valor es necesario para acceder a un determinado dispositivo lógico desde la librería Vesta si no se quiere utilizar el formato *controladora[slot]*. El sistema no impide que existan nombres duplicados, pero al acceder devolverá uno de ellos aleatoriamente.

Los nombres de los dispositivos figuran en la web de configuración.

remoteTimeout (unsigned char, pos 17)

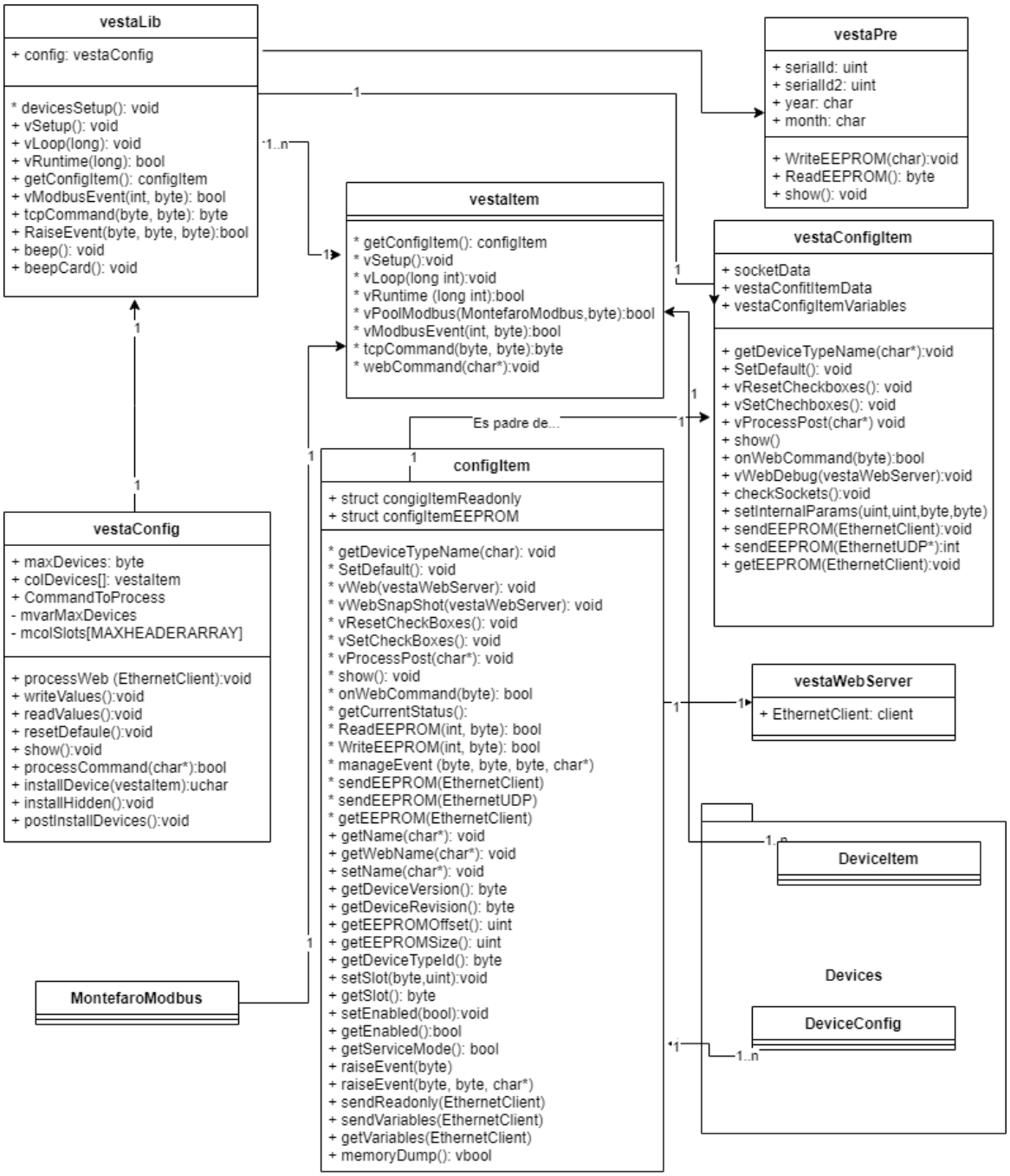
Tiempo (en segundos) que transcurre sin recibir comandos desde el controlador domótico para pasar al modo local.

El valor debería asignarse en función de la latencia máxima de la red local ya que el controlador domótico en un funcionamiento normal envía pulsos UDP a las controladoras para informar de su estado (heartbeats). De esta manera los dispositivos con modo local tomarán el control inmediatamente al detectar la falta del controlador domótico.

Apéndice D

Librería Vesta

Objetos de la controladora



D.1. vestaPre

Librería de cabecera Vesta. Todas las controladoras Vesta almacenan su información de configuración en la memoria EEPROM que permite al módulo comportarse como tal, detectar posibles corrupciones de información y almacenar la información persistente para cada dispositivo.

Las primeras direcciones en la EEPROM definen las características del módulo y sirven a la rutina de inicio para poner en marcha todos los componentes de forma controlada. Esta parte se llama *cabecera* y tiene el siguiente formato:

dirección	dato	tipo	notas
00	32	byte	firma
01	63	byte	firma
02	serialId	unsigned int	nº de serie
04	serialId2	unsigned int	nº de serie 2º
06	month	byte	mes de fabricación [1..12]
07	year	byte	año de fabricación (2 cifras)
08	permission	byte	ver observaciones
09	crc	byte	comprobación de cabecera
10	-	-	comienzo del primer dispositivo

Esta librería sirve para generar una cabecera y también para leerla e interpretarla.
La rutina de inicio de Vesta lee la cabecera en la primera fase de inicialización y si no encuentra los dos primeros valores de la firma (32,63) o la comprobación de cabecera no coincide con el valor calculado dará un mensaje de error en la salida del terminal y activará el led de error interrumpiendo la secuencia y dejando al circuito en un estado de bloqueo.

D.1.1. Componentes de la cabecera

1. firma: Son dos valores constantes (32 y 63) en las dos primeras posiciones de la firma.
2. SerialId: Son dos números enteros sin signo que definen el número de serie de la controladora. El primero es el código del fabricante y el segundo es el número de serie de esta distribución.
   No pueden existir dos controladoras con los dos números de serie idénticos.
3. month: Mes de fabricación. Sólo se admiten valores entre 1 y 12, que ocupan los 4 bits de menos peso del parámetro. Los otros 4 bits deben estar a 0 y están reservados para

futuras prestaciones.

4. year: Año de fabricación. Valor entre 00 y 255 siendo 00 el año 2000 y 255 el año 2255.

5. permission: Es un valor que se calcula operando el número máximo de dispositivos permitidos con el valor del CRC calculado hasta el momento por medio de una suma exclusiva XOR. El número máximo de dispositivos permitidos impide declarar más dispositivos que los establecidos en este valor.
   Por ejemplo, si el número máximo es 10 y se intentan definir 20 dispositivos, la rutina de inicio sólo reconocerá los 10 primeros ignorando el resto.

6. crc: Suma de comprobación. Es un valor que se genera en el momento de almacenar la cabecera en la EEPROM y que se calcula al leerla. Si el cálculo de crc de la rutina de inicio no coincide con la suma de comprobación guardada se interrumpirá la carga y se encenderá el indicador de error.
   Previene errores catastróficos por corrupción de datos y el uso malintencionado de la controladora.

D.1.2. Funciones

WriteEEPROM

```
void WriteEEPROM(unsigned char maxDevices);
```

Formatea la cabecera asignando el máximo de dispositivos con el valor del parámetro maxDevices.
Esta función sólo está disponible en una herramienta de uso interno para formatear las nuevas controladoras. En función de la licencia de uso será necesario solicitar tarjetas Arduino convenientemente formateadas para poder utilizarlas como controladoras Vesta.

ReadEEPROM

```
unsigned char ReadEEPROM ();
```

El resultado de la función puede ser:

- 255: Error de CRC. La controladora quedará bloqueada y se encenderá el testigo de error.
- 0: Error desconocido. Los mismos síntomas.
- otro valor: El número máximo de dispositivos permitidos. La controladora se inicia normalmente y se detiene la carga de dispositivos cuando el índice actual supere este valor.

show

```
void show();
```

Vuelca información sobre la cabecera en la salida del puerto serie.

D.2. vestaBasic

Es un objeto formado con enumeraciones y métodos estáticos de utilidad en el resto de módulo de la librería.

D.2.1. Enumeraciones

TRequest

Lista de posibles comandos desde el controlador domótico (*Lares*).

mnemónico	valor	significado
TCRequestCatalog	0	Catálogo de dispositivos
TCRequestOnlyRead	1	Información de solo lectura a Lares
TCRequestEEPROM	2	Parámetros EEPROM a Lares
TCRequestDynamic	3	Parámetros dinámicos a Lares
TCSendEEPROM	4	Lares escribe parámetros EEPROM
TCSendDynamic	5	Lares escribe parámetros dinámicos
TCCommand	6	Comando hacia un dispositivo Vesta
TCEnd	255	Petición de finalización de conexión

VestaDeviceTypeId

Lista de tipos definidos de dispositivos. Esta lista es susceptible de ampliaciones en sucesivas versiones de la librería.

mnemónico	valor	significado
VDTController	0	Controladora Vesta
VDTInput	1	Entrada binaria
VDTOutput	2	Salida binaria
VDTDHT	3	Módulo climático
VDTGate	4	Puerta o persiana
VDTRfid	5	Lector de tarjeta rfid
VDTRAnalogicInput	6	Entrada analógica
VDTRAnalogicOutput	7	Salida analógica
VDTParking	128	Módulo combinado de parking
VDTElevator	129	Módulo combinado de ascensor
VDTCustom	0xFD	Módulo de desarrollo a medida
VDTTemplate	0xFE	Plantilla de dispositivo
VDTNull	0xFF	Dispositivo nulo

TVestaLibCommand

Comandos internos a procesar por el sistema.
Estos comandos se almacenan en la variable **CommandToProcess** de la instancia de **vestaConfig**. El sistema los procesa en la llamada a **vLoop** del controlador comprobando primero el valor y si es distinto de *TVnull* lo procesa y luego restaura este valor dejando el sistema preparado para el siguiente comando.

mnemónico	valor	significado
TVnull	0	Comando nulo. No hace nada.
TVWriteEEPROM	1	Guarda el estado actual en EEPROM.
TVReboot	2	Fuerza un reinicio de la controladora.
TVBeep	3	Hace un pitido.
TVSetDefault	4	Restaura los valores EEPROM al estado de fábrica.

D.3. vestaConfig

La única instancia de esta clase por controladora contiene todos los dispositivos lógicos definidos y se encarga de acceder a ellos, mostrar su configuración en el entorno web y cargar sus valores iniciales en la rutina de arranque del sistema.

D.3.1. Miembros

maxDevices (unsigned char)

Es el número de dipositivos instalados.

colDevices (vector de punteros a vestaItem)

Contiene las referencias a los dispositivos.
El sistema itera estos objetos en las rutinas de actualización.

colIrqInputs (vector de char con 4 elementos)

Contiene los índices de hasta 4 dispositivos que reaccionan a las interrupciones hardware.
En las iteraciones de evento el sistema se salta estos cuatro índices porque la actualización es responsabilidad de las respectivas rutinas de interrupción.

CommonHeader (vestaConfigItem)

Instancia del contenedor de configuraciones de los dispositivos. Este objeto contiene las referencias a los dispositivos y CommonHeader las referencias a sus respectivas configuraciones.

CommandToProcess (unsigned char)

Siguiente comando que debe procesar el sistema. Este valor se asigna desde el motor web, desde la consola del puerto serie, a través de Modbus o desde el controlador domótico.
Comandos definidos:

- (1) `COMMAND_WRITE_EEPROM`: Escribe en la EEPROM los valores de configuración actuales que no coincidan con los ya almacenados.
- (2) `COMMAND_SET_DEFAULTS`: Escribe en la EEPROM los valores de configuración de fábrica para todos los dispositivos.
- (3) `COMMAND_REBOOT`: Fuerza un reinicio de la controladora.

EEPROMTop (unsigned int)

Es la dirección siguiente a la última ocupada por la configuración del último dispositivo instalado.

D.3.2. Funciones

processWeb

```
void processWeb(EthernetClient *rhs);
```

Recibe la petición desde el cliente http y proporciona la respuesta correspondiente.
Atiende las peticiones GET y POST desde navegadores. Puede mostrar el menú principal, el de cualquier dispositivo presente o realizar cambios en el sistema de acuerdo al contenido de POST.

writeValues

```
void writeValues();
```

Escribe la configuración actual de todos los dispositivos en la EEPROM.
El objeto recorre todos los dispositivos instalados escribiendo sus respectivas secciones EEPROM en la memoria persistente. El algoritmo está optimizado para no repetir escrituras si los datos existentes son idénticos minimizando la posibilidad de agotar los ciclos de escritura disponibles.

readValues

```
bool readValues();
```

Carga en los dispositivos presentes la configuración actual de la EEPROM.
Si se produce un error al cargar o los códigos de comprobación cíclica no coinciden devolverá *False*. El sistema invoca esta función en la rutina de inicio cuando se recupera la alimentación o tras un comando de reset. En caso de fallo de carga, la controladora quedará bloqueada y se encenderá el indicador de error.

resetDefault

```
bool resetDefault();
```

Escribe en la EEPROM los valores asignados de fábrica para cada dispositivo. Devolverá *False* si se produce algún fallo en el proceso.

show

```
void show();
```

Muestra todos los datos de configuración actuales por la salida del puerto serie. Se usa con propósitos de depuración.
Ejemplo parcial del resultado de la llamada a la función *show()*:

```
Configuration Dump:
-Module 0 :
Common
Pruebas
Device Id: 0 V1.6 slot:0 (Enabled)
Raise internal events
Raise UDP events
Manage internal events
Name: Pruebas
Local Ip: 192.168.2.11
Mask: 255.255.0.0
Gateway: 192.168.6.1
Remote IP's:
-0 255.255.255.255
-1 0.0.0.0
-2 0.0.0.0
-3 0.0.0.0
## Main ######################
Sound ENABLED
Watchdog ENABLED
Ethernet Reset ENABLED
Ethernet reset time: 15 minutes.
## HTTP ######################
Service ENABLED
Port 80
## Lares ######################
Service ENABLED
Port 5169
## Modbus ######################
Service disabled
Port 502
## Beat ######################
Service ENABLED
Port 5148
Timer: 30s
## Events######################
Service ENABLED
Port 5168
```

```
-Module 1 :
Abierto
Device Id: 2 V1.0 slot:1 (Enabled)
Raise internal events
Output pin:26
Inverted output
Timer of 2304 seconds.
-Module 2 :
rrado
Device Id: 2 V1.0 slot:2 (Enabled)
Remote mode
Manage internal events
Auto local mode
Output pin:27
Direct output
-Module 3 :
celula
Device Id: 2 V1.0 slot:3 (Enabled)
Service mode
Remote mode
Raise internal events
Manage internal events
Auto local mode
Output pin:28
Inverted output
```

processCommand

```
bool processCommand(const char *rhs);
```

Ejecuta una orden desde el terminal del puerto serie.
Para conocer la lista de posibles comandos y su uso se debe consultar el capítulo correspondiente en el apéndice.

installDevice

```
unsigned char installDevice(vestaItem *element);
```

Añade el dipositivo que se pasa por referencia en el parámetro a la lista de dispositivos gestionada por este objeto. El sistema invoca esta instrucción por cada dispositivo a instalar hasta llegar al máximo permitido por la cabecera.

installHidden

```
void installHidden();
```

Marca el instalador de forma que los dispositivos que se instalen a continuación con *installDevice* no aparecerán en el menú principal ni se verán en el controlador domótico.
Los dispositivos ocultos se utilizan como parte de los dispositivos complejos cuando su operación por separado no interesa por motivos de integridad o para no sobrecargar al usuario con información redundante o innecesaria.

postInstallDevices

```
void postInstallDevices();
```

Rutina que el programa de inicio de la distribución debe llamar obligatoriamente tras haber dado de alta todos los dispositivos. Sirve para realizar tareas de configuración que requieren la presencia de la configuración completa, pero antes de que ésta comience a funcionar.
Por ejemplo, en esta función se realiza la asignación de vectores de interrupción en aquellos dispositivos que han sido marcados para responder ante interrupciones de hardware.

xxtcpSendDevicesInfo

```
void xxtcpSendDevicesInfo(EthernetClient *rhs);
```

Envía al cliente ethernet la enumeración de todos los dispositivos visibles instalados. Se emplea para rellenar el menú principal de selección.

D.4. vestaWebServer

Los módulos **Vesta** implementan un servidor Web que sirve para consultar el estado de los dispositivos y la controladora, permitiendo la edición y modificación de los parámetros, así como la ejecución de algunos métodos. Para todo ello se usa un servidor elemental basado en el framework **Bootstrap**.

D.4.1. Implementación

El controlador Vesta tiene una instancia de esta clase a la que se le pasa un puntero de la clase **EthernetClient** en el que construirá la web y de la que leerá sus comandos.
Contiene 92 funciones que se corresponden con los posibles objetos que puede mostrar el framework, algunos de ellas con parámetros sobrecargados para ofrecer diferentes posibilidades de

invocacion.
Los objetos que heredan de **ConfigItem** hacen uso de estas funciones en las llamadas a los métodos *vWeb* y *vWebSnapshot*.

D.5. vestaLib

La controladora Vesta es un objeto definido en un archivo arduino *extensión .ino* que hereda necesariamente del objeto **vestaLib** y extiende sus métodos virtuales.

D.5.1. Funciones

Constructor

Asigna los valores de versión, anula el próximo comando a procesar e inicia toda la estructura antes de cargar los dispositivos del sistema.

vSetup

```
void vSetup();
```

Es la función de arranque del sistema. El controlador la invoca una sola vez justo después de la llamada al constructor y antes de pasar al modo normal.
Durante esta llamada el controlador realiza estas funciones:

1. Carga los dispositivos.
2. Inicia los registros modbus.
3. Carga los valores de configuración almacenados en la EEPROM.
4. Mapea las interrupciones de hardware (si las hay).
5. Inicia el módulo de Ethernet.
6. Inicia la rutina de servicio *Watchdog* para prevenir fallos de bloqueo.
7. Invoca las rutinas de inicio de cada dispositivo por orden de slot.
8. Provoca el evento *onInit*.
9. Informa que el arranque ha sido correcto por el puerto serie.

devicesSetup

```
virtual void devicesSetup();
```

El controlador debe sobreescribir este método para dar de alta todos los dispositivos que contiene. El sistema invoca este procedimiento en la primera fase de la rutina de inicio.

vLoop

```
void vLoop(long int counter);
```

El sistema llama a vLoop desde la función Loop de Arduino del programa principal. El parámetro *counter* se ignora. En cada llamada de vLoop, el controlador realiza estas funciones:

1. Incrementa un contador de ciclos.
2. Hace lucir el led de actividad.
3. Procesa las interrupciones hardware.
4. Refresca el Watchdog.
5. Responde a los comandos del terminal serie.
6. Responde a los comandos del protocolo tcp de la librería Lares.
7. Responde a los comandos modbus.
8. Responde las peticiones web pendientes.
9. Comprueba que el controlador está habilitado (*enabled* == *True*);
10. Ejecuta la rutina vLoop de cada dispositivo pasándole el contador de ciclos como argumento.
11. Finalmente, en todo caso, llama a *vRuntime* y apaga el led de actividad.

Es importante tener en cuenta que los pasos 5 a 10 son condicionales y dependen del estado de las solicitudes pendientes. Esto quiere decir que si existe algún comando del terminal serie pendiente de ejecutar, el programa lo procesará y luego invocará a vRuntime sin atender el resto de los pasos.
Esta forma de funcionar marca unas prioridades en las que la rutina vLoop de cada dispositivo está en el último lugar. Dicho de otro modo: se ejecuta vLoop de cada dispositivo si no hay comandos pendientes del terminal serie, si no hay peticiones tcp, si tampoco hay peticiones modbus ni peticiones de la web de configuración y el controlador está habilitado.
En la práctica sólo sufriría inanición la rutina vLoop en caso de saturación de peticiones al dispositivo y es una situación que no se dará en el funcionamiento normal.

vRuntime

```
bool vRuntime(long int counter);
```

Procesa los accionamientos activos del módulo. La rutina tiene los siguientes pasos:

1. Comprueba que no está activo el modo de servicio del controlador. (*service == False*). Si no se cumple termina la rutina.
2. Procesa las posibles interrupciones hardware (nuevamente).
3. Refresca el Watchdog.
4. Comprueba si se ha producido algún cambio en el estado de los dispositivos. Si no termina la rutina.
5. Comprueba si existe algún dispositivo pendiente de ejecutar una acción.
6. Si existe ejecuta esa acción, devuelve *True* y termina.
7. En otro caso termina devolviendo *False*.

El número de slot de los dispositivos marca la prioridad para ejecutar sus acciones. Si existen dos dispositivos con números de slot x e y siendo $x > y$, se ejecutará el código de x dejando para la siguiente iteración el código de y, salvo que en la siguiente iteración surja un dispositivo con número de slot z tal que $z < y$.

En la práctica sólo se producirá inanición si un dispositivo está consumiendo los ciclos para realizar acciones de forma reiterada, que en esta estructura es un signo claro de error lógico en ese código.

getConfigItem

```
virtual configItem *getConfigItem();
```

El objeto **configItem** es el contenedor de la configuración del módulo y depende de la distribución. Por ello es responsabilidad del desarrollador de la distribución su creación y este método sirve para que el objeto base pueda acceder a él.

vModbusEvent

```
bool vModbusEvent(int address, uint8_t command);
```

En caso de producirse una llamada modBus el sistema llamará a esta función que debe ser sobrescrita por la controladora. Los parámetros hacen referencia a la dirección y al comando modbus. Debe devolver *True* si existe respuesta definida para el evento o *False* en caso contrario.

tcpCommand

```
uint8_t tcpCommand(uint8_t commandId, uint8_t param);
```

El desarrollador debe sobreescribir este método para gestionar comandos en respuesta a peticiones tcp. Los parámetros son el identificador del comando y un parámetro opcional, a lo que se responderá 0 si el comando ha sido correcto o cualquier otro valor indicando el error producido.
En caso de no sobreescribir la función la implementación base devuelve 255 que es el identificador del error desconocido.

RaiseEvent

```
bool RaiseEvent
(unsigned char slotId,
unsigned char eventId,
unsigned char paramLength,
unsigned char *buffer,
bool internal,
bool external);
```

Rutina invocable desde los dispositivos instalados para provocar eventos. Los parámetros son:

- slotId: Número de slot del dispositivo que desencadena el evento.
- eventId: Código del evento.
- paramLength: Longitud (en bytes) del parámetro (o cero si no hay).
- buffer: Puntero a una dirección de memoria donde está el contenido del parámetro.
- internal: *True* si el evento se desencadena de forma interna (hacia el resto de dispositivos de esta misma controladora).
- external: *True* si el evento se desencadena en forma de paquete UDP para su propagación por ethernet.

El sistema se encargará de procesar la petición encaminando el evento por el cauce adecuado. Se puede provocar un evento que sea sólo interno, sólo externo, interno y externo o ninguno de los dos, con lo que simplemente se ignorará el comando.

beep

```
void beep();
```

Hace un pitido por el zumbador de la controladora.
Se puede usar desde cualquier dispositivo instalado y desde la propia controladora para señalizar cualquier situación más allá de las previstas por defecto.

beepCard

```
void beepCard();
```

Similar a la anterior, el tono es diferente. Está pensado para indicar que la controladora ha reconocido una identificación, como por ejemplo al leer con éxito una tarjeta rfid, aunque podría servir igualmente ante activaciones de detectores de presencia, teclados de contraseñas, etc.

renderWebIndex

```
virtual void renderWebIndex(EthernetClient *client);
```

Función virtual que la controladora debe sobreescribir para mostrar un menú principal web personalizado para la distribución.
Se puede mantener la implementación genérica del objeto base cuyo código se muestra a continuación:

```
void vestaLib::renderWebIndex(EthernetClient *rhs)
{
int auxCol=0;
vestaWebServer dd = vestaWebServer(rhs);
dd.beginContainer();
dd.beginRow();
for(int i=0;i<config.maxDevices;i++)
{
if(auxCol>3)
{
dd.endRow();
dd.beginRow();
auxCol=0;
}
dd.beginCol();
config.colDevices[i]->getConfigItem()->vWebSnapshot(&dd);
auxCol++;
dd.endCol();
}
dd.endRow();
dd.endContainer();
}
```

Utiliza un objeto vestaWebServer para rellenar la página con una lista de los dispositivos instalados ordenados por columnas de cuatro en cuatro, mostrando su vista reducida *vWebSnapshot*.

getControllerModel

```
virtual void getControllerModel(const char *destination);
```

Escribe en *destination* el nombre de la distribución. Este identificador aparecerá para identificar el modelo de controladora en los menús de configuración web, en el controlador domótico y en el terminal serie.
Por ejemplo, en la distribución *Model A* el código de remplazo en el objeto principal será:

```
void nombreClase::getControllerModel(const char *destination)
{
sprintf(destination,"Model A");
}
```

auxPoolSerial

```
void auxPoolSerial();
```

Lee la entrada desde el terminal serie y procesa los comandos. En la versión original de la clase el comportamiento y las instrucciones que puede procesar son las reflejadas en este texto, aunque el desarrollador puede expandirlo a voluntad con nuevos comandos. Esto se hace remplazando el método original por uno nuevo.

Objetos de cada dispositivo

D.6. VestaItem

Es la clase base de todos los dispositivos Vesta. En ella queda definido el comportamiento con las acciones que realizará, los eventos que provoca o recibe y las actividades del modo automático.

D.6.1. Funciones

getConfigItem

```
virtual configItem *getConfigItem();
```

Devuelve una referencia al objeto que almacena la configuración de este dispositivo.
Cada dispositivo tiene que tener su respectivo contenedor de configuración *configItem*.

vSetup

```
virtual void vSetup();
```

Rutina de inicio. El sistema llama a esta función una sola vez por cada dispositivo al conectar la alimentación o tras cada pulsación de reset.

vLoop

```
virtual void vLoop(long int counter);
```

Rutina de pooling. El sistema llama a esta rutina para todos los dispositivos cuya entrada de habilitación **enable** esté activada. Durante esta función el dispositivo se encargará de actualizar las entradas, **pero no las salidas ni los métodos automáticos**.
El parámetro *counter* contiene el valor del temporizador general del controlador en microsegundos y sirve para gestionar las rutinas de temporización sin tener que emplear una variable dedicada. Si la actuación del módulo no requiere conocer el tiempo transcurrido se puede ignorar el valor.

vRuntime

```
virtual bool vRuntime(long int counter);
```

Rutina de ejecución. El sistema llama a esta rutina para todos los dispositivos cuya entrada de servicio **service** esté desactivada (*False*) y cuando la habilitación **enable** esté activada (*True*). En esta rutina el objeto actualizará sus salidas y realizará todas las actividades automáticas que correspondan al estado activo.

vPoolModbus

```
virtual bool vPoolModbus(MontefaroModbus *rhs, uint8_t deviceIndex);
```

Ante una llamada del subsistema ModBus, el controlador invoca esta función en todos los dispositivos para que actualicen la salida de datos. Es equivalente a la función *vLoop* por lo que se tendrá en consideración el valor del flag **enable**.
Los parámetros son una referencia al objeto modbus y el número de slot afectado por la iteración.

vPoolModbus

```
virtual bool vModbusEvent(int address, uint8_t command);
```

Ante una llamada del susbistema Modbus, el controlador invoca esta función en todos los dispositivos que no estén en modo servicio (**service**). El dispositivo afectado por el comando realizará las funciones requeridas por la orden modbus. Los parámetros son la dirección modbus requerida y el comando ejecutado.

tcpCommand

```
virtual uint8_t tcpCommand(uint8_t commandId, uint8_t param);
```

Ejecuta un comando desde el controlador domótico. Los dos parámetros se corresponden con el identificador del comando y un parámetro opcional, ambos de tipo unsigned char. Estos valores dependen del tipo de dispositivo que sea (a consultar en esta documentación) y pueden devolver una respuesta también de tipo unsigned char.

webCommand

```
virtual void webCommand(const char *rhs);
```

Ejecuta un comando desde la página web de configuración. El id del comando y los posibles parámetros dependen de cada tipo de dispositivo y la respuesta (de haberla), se mostrará en la misma web.

D.7. ConfigItem

El funcionamiento de cada dispositivo **Vesta** queda definido por una lista de parámetros que diferencian unos de otros. Estos parámetros se dividen en tres tipos:

1. Fijos: Se establecen de fábrica para cada dispositivo y no se pueden cambiar.
2. EEPROM: Se pueden modificar y quedan guardados incluso cuando el circuito pierde la alimentación.
3. Variables: Se pueden modificar numerosas veces, sirven para recordar información de uso frecuente y no se almacenan en las pérdidas de alimentación.

Todos estos valores residen en diferentes zonas de la memoria del controlador y el objeto **ConfigItem** es el encargado de acceder a ellos y sirve de puente entre el usuario, el circuito y los

propios dispositivos físicos.
Cada familia de dispositivos lógicos tiene su propio objeto de configuración que hereda de **ConfigItem**. A su vez, el objeto raíz contiene unos parámetros de configuración que son comunes a cuaquier dispositivo lógico.

D.7.1. Funciones

getDeviceTypeName

```
virtual void getDeviceName (char *destination)=0;
```

Devuelve el nombre de este tipo de dispositivo copiándolo en el array del parámetro. El valor devuelto se muestra en la salida del terminal serie y en la web de configuración del controlador.

SetDefault

```
virtual void SetDefault();
```

Aplica los valores por defecto. Se usa al iniciar el controlador y a petición del usuario desde el terminal serie, la web de configuración o desde el módulo de gestión domótica.

vWeb

```
void vWeb(vestaWebServer *rhs);
```

Muestra los valores de configuración actuales en la web de configuración a través de la instancia vestaWebServer que se pasa por referencia.

vWebSnapshot

```
virtual void vWebSnapshot(vestaWebServer *rhs);
```

Muestra una versión resumida de los parámetros más básicos de la configuración. Esta función se usa en las páginas que ofrecen un resumen del estado actual de varios dispositivos en formato lista o tabla. Es importante moderar el tamaño y la cantidad de proceso en las llamadas a esta función porque pueden degradar la respuesta de la interface.

vResetCheckboxes

```
virtual void vResetCheckboxes();
```

Los campos de tipo boolean se muestran con controles de tipo checkbox. La forma de gestionarlos requiere almacenar temporalmente estos valores y empaquetarlos en variables numéricas que ocupan menos en la limitada memoria del controlador. Esta función y la siguiente son necesarias para gestionar las funciones que se ocupan de ello.

vSetCheckboxes

```
virtual void vSetCheckboxes();
```

Establece los valores de los checkboxes que se muestran en la web de acuerdo al estado actual de las variables de tipo booleano que contiene el dispositivo.

vProcessPost

```
virtual void vProcessPost(char *rhs);
```

Lee la entrada de datos del usuario en la vista web para hacer los correspondientes cambios en los parámetros del dispositivo. La propia clase ya contiene el código auxiliar necesario para hacer este trabajo.

Show

```
virtual void Show();
```

Vuelca toda la información sobre este dispositivo en la salida del terminal de texto por el puerto serie. Se recomienda seguir un estilo similar al de los dispositivos ya creados.

onWebCommand

```
virtual void onWebCommand(uint8_t rhs);
```

Procesa un comando desde la web:
Los dispositivos pueden ser de entrada y también de salida. Algunos de ellos pueden realizar algún trabajo como encender y apagar una luz o un motor. Aunque se han diseñado para recibir órdenes del controlador domótico, también existe la posibilidad de ejecutar alguno de estos comandos desde la propia página de configuración web haciendo click en un botón. En este caso el sistema llamará a esta función pasándole como parámetro el identificador del botón que fue pulsado.

WriteEEPROM

```
virtual bool WriteEEPROM(unsigned int &index, uint8_t &crc);
```

Escribe los valores actuales en la memoria EEPROM a partir de la dirección de comienzo accesible en **getEEPROMOffset()**, procesando tantas direcciones como indica el valor de **getEEPROMSize()**.
El parámetro **index** es un valor por referencia que se actualiza a la última dirección que se escribió y el parámetro **crc** es un contador cíclico de comprobación que sirve para comprobar la integridad de los transferidos en la próxima lectura.

ReadEEPROM

```
virtual bool ReadEEPROM (unsigned int &index, uint8_t &crc);
```

Lee los valores desde la memoria EEPROM y los copia en las variables correspondientes del dispositivo.

raiseEvent

```
void raiseEvent(unsigned char eventId);
void raiseEvent(unsigned char eventId, unsigned char paramLength,
unsigned char *params);
```

Desencadena un evento con o sin parámetros. Los eventos son llamadas que realiza el dispositivo hacia otros dispositivos de la misma controladora o hacia el sistema domótico. El usuario puede configurar cada dispositivo para que emita eventos de forma local o a través de ethernet.
Los eventos locales se usan en ciertos dispositivos compuestos que se sirven de otros dispositivos más simples. Por ejemplo, el módulo de puertas o persianas motorizadas utiliza dispositivos de entrada para leer los sensores y dos dispositivos de salida para conectar y desconectar los motores de apertura y cierre por medio de eventos internos.
Los eventos externos son la mejor forma de actualizar el estado del controlador domótico sin sobrecargar la red haciendo pooling.

manageEvent

```
virtual bool manageEvent(unsigned char slotId, unsigned char eventId,
unsigned char paramLength, unsigned char *params);
```

La controladora invoca esta función cuando se ha producido un evento en otro dispositivo o bien a través de ethernet y el dispositivo contendrá aquí el código de respuesta.

sendReadonly, sendEEPROM, sendVariables

```
void sendReadonly(EthernetClient *client);
virtual void sendEEPROM(EthernetClient *client);
void sendVariables(EthernetClient *client);
```

Envía los datos de solo lectura al controlador domótico a demanda. Los datos siguen un formato estricto que se envía tal cual reside en la memoria de la controladora. A partir del tipo de dispositivo y de los números de versión y revisión, el controlador domótico debe procesar esta información para extraer de ella los valores correctos.

getEEPROM, getVariables

```
virtual void getEEPROM(EthernetClient *client);
void getVariables(EthernetClient *client);
```

Obtiene los datos desde el controlador domótico cuando el administrador desea configurar el módulo Vesta en remoto.

memoryDump

```
void memoryDump();
```

Escribe en la salida del terminal el contenido de todas las direcciones de memoria asociadas a este dispositivo. Es una función disponible desde la línea de comandos del terminal serie para propósitos de depuración de software.

D.7.2. Funciones de configuración común

Estas funciones afectan a los valores de ciertos parámetros que son comunes a todos los dispositivos y que por tanto, implementarán todas las clases heredadas de ésta.

getName

```
void getName(char *destination);
```

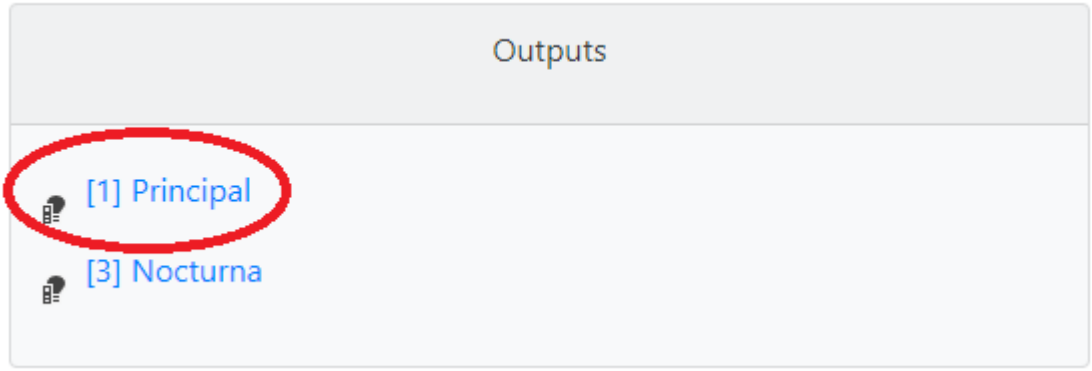
Copia en el array *destination* el nombre de este dispositivo.

> Importante: No confundir con getDeviceTypeName, que devuelve el nombre del tipo del dispositivo. Por ejemplo, si se habla de un controlador de persianas motorizadas, getDeviceTypeName devolvería *Persiana Motorizada* y getDeviceName devolvería *Persiana1* o *Persiana Salón*, según la instancia del dispositivo invocada.

getWebName

```
void getWebName(char *destination);
```

Copia en el array *destination* el nombre de este dispositivo tal como se muestra en la web.



Como se puede apreciar, además del propio nombre se incluye un icono y el número de slot del controlador manteniendo una estructura fácil de identificar al usuario.

setName

```
void setName(const char *origin);
```

Entrada de datos para que el usuario pueda cambiar el nombre de una instancia de dispositivo concreta.

getDeviceVersion, getDeviceRevision

```
unsigned char getDeviceVersion();
unsigned char getDeviceRevision();
```

Devuelve los números de versión y revisión de la especificación de este dispositivo. Estos valores vienen asignados de fábrica y el usuario no puede modificarlos.
Sirven de información al controlador domótico para saber qué comandos pueden estar disponibles o no y cuántos valores de configuración pueden tener dependiendo de la versión. Estos valores son comunes a todos los dispositivos del mismo tipo que residan en el mismo controlador Vesta.

getDeviceTypeId

```
unsigned char getDeviceTypeId();
```

Cada tipo de dispositivo lógico tiene asignado un identificador numérico que el sistema domótico reconoce y gestiona. Existen 254 valores posibles, estando el número 0 reservado para el propio controlador. Los valores entre 1 y 127 corresponden a dispositivos estándar según Vesta y el resto son implementaciones libres para las que hay que añadir el correspondiente código en sus respectivos gestores domóticos.

getEEPROMOffset, getEEPROMSize

```
unsigned int getEEPROMOffset();
unsigned int getEEPROMSize();
```

Devuelven las direcciones de memoria de comienzo y tamaño de la región EEPROM correspondiente a este dispositivo.
Tanto para recuperar los datos EEPROM como para almacenarlos, es necesario conocer con detalle la dirección de comienzo y la longitud de la información. El controlador utiliza estos valores para realizar los accesos a la memoria persistente.

getSlot

```
unsigned char getSlot();
```

Devuelve el sector (slot) de la controladora en que reside el dispositivo. Los números de slot se muestran siempre entre corchetes y sirven como índices de un array para acceder a cada dispositivo. Cada controladora vesta puede tener un máximo teórico de 256 slots, siendo el [0] el asignado para la propia controladora.
El controlador principal domótico accede a cada dispositivo por su número de slot. Si *salon* es el nombre de una controladora y la persiana está configurada en el slot 5, se invocará como *salon[5]*.

setSlot

```
void setSlot(unsigned char rhs,unsigned int EEAddress);
```

La rutina de arranque del sistema invoca esta función una sola vez por dispositivo durante la secuencia de inicio. En cada llamada asigna los valores de slot y la dirección de comienzo en la EEPROM que luego se accede en la función *getEEPROMOffset()*.

getCurrentStatus

```
virtual unsigned char getCurrentStatus();
```

Devuelve un valor numérico que corresponde al icono que muestra el estado actual de este dispositivo en la vista web.

getEnabled, setEnabled

```
bool getEnabled();
void setEnabled(bool rhs);
```

Habilitación del dispositivo: cuando está desactivado se interrumpe el funcionamiento, deja de responder a cambios o eventos y muestra la salida correspondiente al estado de reposo.

getServiceMode

```
bool getServiceMode();
```

Obtiene el valor del modo de servicio. Un dispositivo en modo de servicio responde a eventos y procesa los valores de entrada, pero no realiza ningún cambio en los valores de salida ni ejecuta comandos de forma automática. El usuario puede alterar a voluntad todos los valores de salida desde la web en este modo.

getHardwareInterrupted

```
virtual bool getHardwareInterrupted();
```

El dispositivo admite interrupciones por hardware: Los controladores Arduino permiten ejecutar código de forma instantánea mediante interrupciones ante el cambio de ciertos puertos de entrada (consultar documentación de cada controlador). Al habilitar este modo se anula el procedimiento de detección de cambios por pooling del controlador vesta para este dispositivo asumiendo que será función de la rutina de interrupción asignada.

Apéndice E

Comandos

Todos los módulos Vesta tienen salida a un terminal en modo texto por el puerto serie principal. El terminal se utiliza para monitorizar los procesos internos del controlador con fines de diagnóstico y también para ejecutar órdenes básicas en la fase de configuración.

Existen tres tipos de órdenes: Formateo, asignaciones y comandos.

E.1. Formateo

Es un tipo especial de comando que construye la estructura del controlador iniciando la sección EEPROM. Para formatear un controlador se debe escribir:

```
sys [numero]
```

Donde *[número]* es un valor entre 1 y 255.

Al ejecutarlo, el módulo escribirá la cabecera Vesta con número de serie 0, mes 0, año 0 y permisos para instalar un máximo de dispositivos igual al número introducido.

Importante: tras esta orden se perderá el contenido de todo lo almacenado en la EEPROM y la controladora quedará inutilizable. Sólo se debe usar para iniciar nuevas controladoras en fábrica. Sólo debe hacer uso de esta función el administrador o fabricante.

E.2. Asignaciones

Son sentencias que permiten dar valores a los parámetros principales de configuración. Están pensadas para fabricantes o administradores.

Existen dos posibles asignaciones:

Número de serie

```
set serial [id0],[id1]
```

donde [id0] e [id1] son los nuevos números de serie. Por ejemplo, si se pretende iniciar una controladora con el número de serie 1578 para el fabricante 540, el comando a introducir será:

```
set serial 540,1578
```

Fecha de fabricación

```
set date [month],[year]
```

donde [month] es el número del mes actual (entre 1 y 12) y [year] es el año actual entre 00 (para 2000) y 255 (para el año 2255). Si se pretende iniciar una controladora para junio de 2022 el comando será:

```
set date 6,22
```

E.3. Comandos

Son órdenes para la controladora. Todas ellas comienzan con la sentencia *cmd*.

E.3.1. format

```
cmd format
```

Vacía todo el espacio asignado para la EEPROM sustituyendo los datos existentes por ceros. Se usa al reformatear una controladora tras un uso diferente para evitar que la información previa pueda causar comportamientos erráticos en la nueva controladora Vesta.

Usar este comando eliminará toda la información existente y dejará inútil el dispositivo.

E.3.2. reset

```
cmd reset [slot]
```

donde [slot] es 0 o un número de slot utilizado.

- Si [slot] = 0 restablecerá todos los valores de todos los dispositivos y de la controladora a los ajustes de fábrica de cada dispositivo.
- en otro caso restablecerá el dispositivo especificado a sus valores de fábrica.

E.3.3. load

```
cmd load
```

Carga todos los valores desde la EEPROM. Este proceso se hace de forma automática al iniciar la controladora.

E.3.4. save

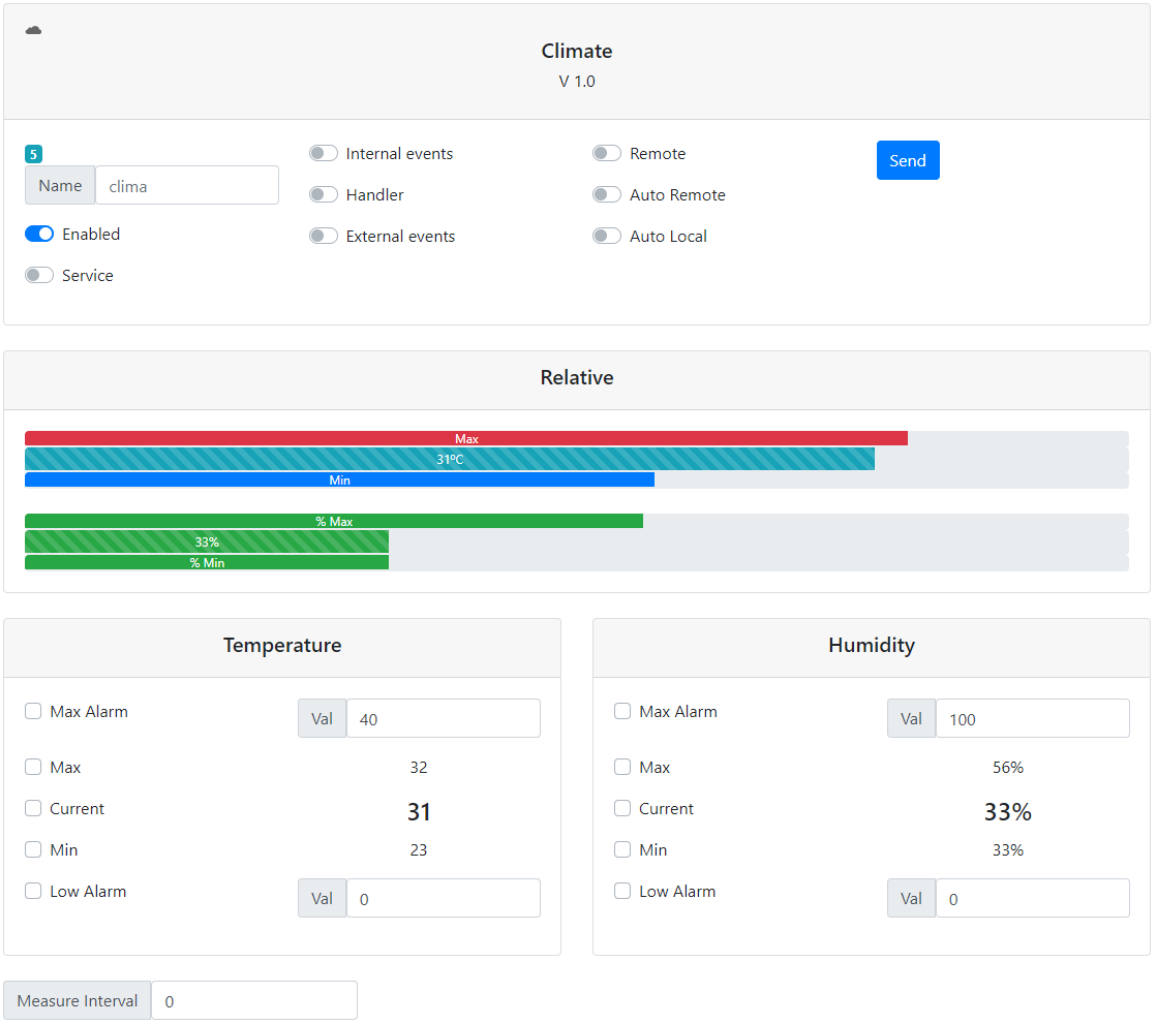
```
cmd save
```

Guarda en la EEPROM los valores actuales de la controladora.

E.3.5. dump

```
cmd dump [slotId]
```

Muestra el contenido de todas las posiciones de memoria para el dispositivo con el número *[slotId]*.
Por ejemplo, en la controladora de la imagen, el dispositivo con el slot 5 es un sensor meteorológico de temperatura y humedad.



Al introducir la orden se tiene la misma respuesta en modo texto desde el terminal serie:

```
>cmd dump 5

VestaConfigItem memory dump
===========================
Device: Climate
Slot: 5
Addrs Rel Value Ascii Bin
0 0 3   00000011
1 1 5   00000101
2 2 1   00000001
3 3 0   00000000
4 4 10   00001010
5 5 40 ( 00101000
6 6 52 4 00110100
7 7 7   00000111
8 8 255 . 11111111
9 9 255 . 11111111
========== EEPROM ==========
============================
```

```
Offset: 10
Addrs EEPROM Rel Value Ascii Bin
10 161 0 1   00000001
11 162 1 99 c 01100011
12 163 2 108 l 01101100
13 164 3 105 i 01101001
14 165 4 109 m 01101101
15 166 5 97 a 01100001
16 167 6 0   00000000
17 168 7 0   00000000
18 169 8 0   00000000
19 170 9 0   00000000
20 171 10 0   00000000
21 172 11 0   00000000
22 173 12 0   00000000
23 174 13 0   00000000
24 175 14 0   00000000
25 176 15 0   00000000
26 177 16 0   00000000
27 178 17 114 r 01110010
28 179 18 0   00000000
29 180 19 0   00000000
30 181 20 144 . 10010000
31 182 21 1   00000001
32 183 22 0   00000000
33 184 23 0   00000000
34 185 24 100 d 01100100
35 186 25 0   00000000
36 187 26 0   00000000
37 188 27 0   00000000
38 189 28 0   00000000
39 190 29 0   00000000
========= variables ========
============================
Offset: 40
Addrs Rel Value Ascii Bin
40 0 230 . 11100110
41 1 0   00000000
42 2 64 @ 01000000
43 3 1   00000001
44 4 64 @ 01000000
45 5 1   00000001
46 6 33 ! 00100001
47 7 0   00000000
48 8 33 ! 00100001
49 9 0   00000000
50 10 56 8 00111000
51 11 0   00000000
```

```
VestaConfigItem memory dump
===========================
Device: Climate
Slot: 5
Addrs   Rel     Value   Ascii   Bin
0       0       3               00000011
1       1       5               00000101
2       2       1               00000001
3       3       0               00000000
4       4       10              00001010
5       5       40      (       00101000
6       6       52      4       00110100
7       7       7               00000111
8       8       255     ?       11111111
9       9       255     ?       11111111
========== EEPROM ==========
============================
Offset: 10
Addrs   EEPROM  Rel     Value   Ascii   Bin
10      161     0       1               00000001
11      162     1       99      c       01100011
12      163     2       108     l       01101100
13      164     3       105     i       01101001
14      165     4       109     m       01101101
15      166     5       97      a       01100001
16      167     6       0               00000000
17      168     7       0               00000000
18      169     8       0               00000000
19      170     9       0               00000000
20      171     10      0               00000000
21      172     11      0               00000000
22      173     12      0               00000000
23      174     13      0               00000000
24      175     14      0               00000000
25      176     15      0               00000000
26      177     16      0               00000000
27      178     17      114     r       01110010
28      179     18      0               00000000
29      180     19      0               00000000
30      181     20      144     ?       10010000
31      182     21      1               00000001
32      183     22      0               00000000
33      184     23      0               00000000
34      185     24      100     d       01100100
35      186     25      0               00000000
36      187     26      0               00000000
37      188     27      0               00000000
38      189     28      0               00000000
39      190     29      0               00000000
========= variables ========
============================
Offset: 40
```

E.3.6. show

```
cmd show pre
cmd show [deviceId]
cmd show
```

Muestra la información de configuración actual.

- pre: Presenta la información de la cabecera.

deviceId , donde *[deviceId]* es el número de un dispositivo instalado: muestra la información de este dispositivo.

- sin parámetros: Enumera todos los dispositivos mostrando sus respectivas informaciones por orden de número de slot.

Por ejemplo, para consultar la información del slot 5:

```
>cmd show 5

Configuration for device no 5:
clima
Device Id: 3 V1.0 slot:5 (Enabled)
Temperature values:
Alarms: (Min=0ºC Max=40ºC
Min/Max (Min=23ºC Max=32ºC)
Last known value: 31ºC
Humidity:
Alarms: (Min=0% Max=100%)
Min/Max (Min=33% Max=56%)
Last known value: 33%
Active events:

Measure interval: 0milliseconds.
Wiring:
Data pin: 7
```

E.3.7. peek, poke

```
cmd peek [address]
cmd poke [address],[value]
```

Consulta o realiza cambios en los valores de las direcciones de la EEPROM. **Precaucion:** Este comando puede dejar la controladora o sus dispositivos en estados críticos en los que podría ser

imposible recuperar el control. Sólo deberá ser manejado por administradores, servicio técnico o personal plenamente consciente de las consecuencias y que tenga conocimiento de lo que está haciendo a nivel técnico.

La instrucción peek muestra el valor de una determinada dirección de la memoria.

```
> cmd peek 5

Address 5 value is 1
```

La instrucción poke permite cambiar este valor.

```
> cmd poke 5,130

Ok

> cmd peek 5

Address 5 value is 130
```

E.3.8. help

```
cmd help
```

Muestra la lista de comandos disponibles para el terminal serie en esta distribución.

```
> cmd help

Settings (set x v):
===================
serial s0,s1 Serial Id (primary,secondary)
date m,y Manufacture date (month,year)
Commands:
=========
sys x Activate system on board (witouth cmd, x is max number of devices).
format Has total empty on EEPROM
reset Reset EEPROM
load load data from EEPROM
save store data to EEPROM
dump x Memory dump for device slot x
edump x EEPROM dump (64bytes) from address x
show [x] Information about x (or all devices if no x)
poke x,y Store value y on EEPROM address x
peek x Show current EEPROM address x value
```

Apéndice F

Anécdotas Domóticas

Los árboles más altos del bosque son los que tienen las raíces más profundas. Tras cada actividad humana, y un trabajo de investigación científica es una más, existe siempre otra historia sin la que no sería posible realizar la principal. De alguna forma siempre hay que pagar un precio por dejar una pequeña huella y en este caso fue convivir en familia con un sistema en evolución, o como diría Thomas Edison, *con las mil formas de tecnología que no deberían llamarse domótica*.

Todos estos años han dado para recoger momentos de crisis, de angustia, de desesperación y, ¿por qué no?, también de humor. No quería dejar pasar la oportunidad de compartir con los lectores de este trabajo esos momentos puramente surrealistas que, de alguna forma, también formaron parte del proceso que desembocó en este punto.

Presencia

La primera instalación era un PC industrial conectado con una controladora PCI que tenía 16 entradas y 16 salidas binarias a relé. Estaba programado en Visual Basic .NET y tenía un método muy básico de respuesta a eventos. Por sacar alguna ventaja, un programa tan sencillo tenía pocas probabilidades de fallar, sin embargo uno de sus funciones era hacer sonar un archivo Wav de campanas cuando la casa tenia constancia de presencia humana en el interior y algún otro humano pulsaba el botón del timbre situado en el patio comunitario, que es por donde se entra a la vivienda.

Pues bien... tras unos meses funcionando como se esperaba el timbre comenzó a sonar *solo*. Cuando digo *solo* quiero decir que no había nadie en el patio y que las campanas sonaban. Me volví loco buscando el error en el programa y no pude. La zona a revisar era simple. Luego inspeccioné el cableado por si hubiera cualquier derivación, lo renové... y el timbre seguía sonando cuando menos nos lo esperábamos. Tuve que descartar fallos de software porque las campanas nunca sonaron a horas imposibles de la noche (salvo cuando alguien llamaba de verdad). No... el timbre solía sonar justo cuando menos atención prestábamos al exterior. Era imposible pulsar el botón y escapar antes de que pudiéramos asomarnos.

Al final nos tomamos el fenómeno con humor y bautizamos al misterioso visitante como *presencia*. El timbre dejó de funcionar a finales del año 2008 cuando Lorena supo que esperaba un hijo y no volvió a hacerlo más.

La casa de la basura

Tenemos en el pueblo un servicio municipal de recogida selectiva de basura. Cada día toca sacar un tipo diferente de basura (envases, orgánica, cartones, etc.). Se nos ocurrió aprovechar la domótica para que el procesador central nos avisase con voz sintética del tipo de basura que tocaba sacar a la hora de la recogida. Era práctico.

Tras unos años de funcionamiento conocimos a los vecinos nuevos, que llevaban con nosotros apenas semanas, o quizá meses. Coincidió que mientras hablábamos la casa hizo su anuncio del día y entonces la vecina dijo con naturalidad: *Ah. Vosotros sois los de la casa de la basura.*

Había sonado mal, así que matizó sus palabras. Nos contó que los avisos se escuchaban desde su casa y que también les servían a ellos para recordar lo que tenían que sacar y por eso nuestra casa era eso, ¡la de la basura!

Las persianas que dejaron de funcionar

Uno de los elementos más espectaculares de la casa son las persianas motorizadas del salón. Vivimos en Mallorca y aquí una persiana es una contraventana con lamas que dejan pasar un poco de luz al través, así que el motor de las persianas pliega y despliega las hojas hacia el exterior como lo harían las contraventanas. Al menos así era hasta que una mañana el coche de un vecino aparcó pegado a la fachada. Cuando hacía frío en el salón, el sistema despliega las persianas para que entre luz y ayude a elevar la temperatura antes de encender la calefacción y como en Mallorca amanece una hora antes que en la península, hay claridad antes de despertar.

Pero esa mañana la hoja de la persiana topó con el coche aparcado y la electrónica que traía (creada por personas que no contemplaban esa posibilidad), no lo desconectó haciendo que ambos, motor y controlador se quemasen. Entonces nos quedamos sin poder abrir esa ventana (la mitad de la luz).

El módulo de apertura de puertas y ventanas de Vesta incorpora una entrada de fin de carrera o sobrecarga que interrumpe la apertura al detectar un obstáculo y tiene en cuenta el desfase con respecto a lo previsto para proceder al cierre, de forma que en la operación inversa la otra hoja inicia su recorrido y luego se pone en marcha un temporizador exactamente igual al tiempo que transcurrió hasta la detección del obstáculo de la primera hoja. Entonces conecta el motor y el cierre se realiza perfectamente sincronizado.

Incendio en la terraza

Estaba en casa trabajando con el ordenador cuando sentí ruidos extraños en la terraza donde tenemos la lavadora, la secadora y el termo. No le di más importancia hasta que la domótica comenzó a emitir mensajes sin sentido. Al levantarme para ver lo que pasaba en el armario eléctrico vi las llamas por la ventana de la terraza y logré apagar el fuego. Luego supe que la

culpa la tuvimos por comprar la secadora. El enchufe podía aguantar el consumo de la lavadora y el del termo del agua, pero ya era demasiado si además estaba funcionando la secadora. Ese día coincidieron los tres, el cable se sobrecalentó y... bueno, aprendí que una buena instalación domótica debe tener detectores de incendios y medidores de intensidad eléctrica.

Expertos en el escritorio remoto de Windows

La última versión del controlador central antes de Lares se basaba en un servicio ASP MVC que estaba preparado para correr en un contenedor de Docker, pero que con propósito de agilizar las pruebas acabó ejecutándose en una ventana del terminal de texto en un pequeño ordenador Intel NUC al que accedemos con el escritorio remoto cuando hay problemas. En esta versión todos los controladores, incluidos los Vesta, únicamente funcionan en modo remoto, por lo que cuando *cae* la red o sucede algún problema con el programa principal, la domótica de la casa entera deja de funcionar. Esto puede resultar hasta divertido si ocurre un domingo a mediodía, pero resulta frustrante cualquier día de diario en la madrugada o al llegar tarde a casa tras un día de trabajo, porque hay que abrir sesión en algún ordenador de la casa, ejecutar el escritorio remoto, abrir una ventana de comandos y ejecutar el servicio.

Incluso cuando no falla nada, hay que abrir el dichoso escritorio remoto, porque Microsoft ha adquirido la molesta manía de forzar la actualización automática de sus sistemas y, aunque el servicio está programado para su ejecución al iniciar el sistema, no se ejecuta hasta abrir sesión. Seguro que existe alguna forma de configurarlo, se sabe que volviendo a dejar el programa como un servicio se resolvería el problema, pero requiere perder una tarde en ello y hemos asumido que Lares será un servicio y que cuando caiga, todas las controladoras Vesta pasarán al modo remoto.

Mientras tanto, los que vivimos en esta casa nos hemos vuelto expertos en sesiones remotas.

El termo del agua caliente

He dejado para el final la más dolorosa de las aventuras. Ocurrió con la segunda versión de la domótica centralizada, cuando los precios de la luz eran asequibles y la única complicación era jugar con dos periodos tarifarios del día. El controlador desconectaba el termo durante la franja *cara* y lo conectaba a partir de medianoche. En invierno solemos ducharnos una vez al día; yo por la mañana, para despejarme y Lorena por la noche, para relajarse antes de dormir. El sistema domótico tenía una entrada directa del termostato que le permitía detectar cuánto tiempo había estado calentando la resistencia e incluso podía evaluar si estábamos en casa o no, basándose en que cuando se calienta el agua para compensar la pérdida térmica por defectos del aislamiento se tardan escasos minutos en volver a la temperatura que detiene el termostato.

En condiciones normales el sistema funcionaba bien y era eficiente. El termo tenía capacidad suficiente para mantener caliente el agua para los dos turnos de ducha del día. Sin embargo todavía no había aprendido nada de las complicaciones que supone gestionar un mecanismo eléctrico con existencia física real. Ocurrió que un día, en lo más frío del invierno, quizá por culpa

de una sobrecarga debida a una tormenta, cayeron varios de los disyuntores magnetotérmicos de la instalación y entre ellos el del termo. Por despiste, nadie lo repuso y cuando la domótica fue a conectar el calentador del termo la resistencia no se encendió. El sistema lo interpretó como el final del proceso de calentamiento, así que no hizo ningún aviso. El primer día me duché por la mañana, ella lo hizo por la noche y el agua salió tan caliente como siempre. El segundo día me duché por la mañana con el agua un poco más templada, pero no noté nada especialmente molesto. Esa noche, cuando ella se fue a duchar, tras un pequeño inicio templado que permitió que pudiera enjabonarse, le obsequió con un chorro de agua helada que procedía de algún manantial de la montaña...

Explicar la bronca que me cayó esa noche daría para redactar varios libros de terror.

Lo triste de la anécdota es que, aunque aprendí mucho de aquel fallo, volvió a ocurrir al menos un par de veces más.

Esto tiene que servir para ilustrar la conclusión final: La domótica debe asumir el control de una vivienda real, sometida a desgaste, compuesta por aparatos reales, sometidos a desgaste y es totalmente imposible evitar que algo falle. Hay que vivir con el error y proporcionar siempre alternativas para cuando ocurre. El usuario final nunca recordará lo espectacular que fue vivir en aquella casa que le adivinaba el pensamiento si acaba duchándose con agua fría un solo día.

Para más información consulte la web del autor (fig F.1) https://www.montefaro.eu

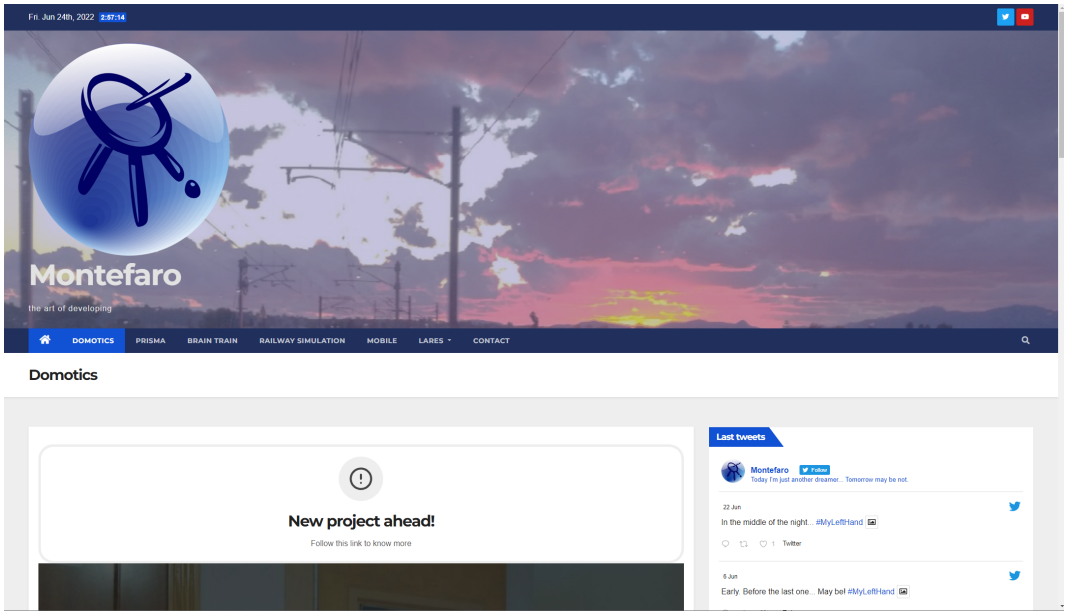


Figura F.1: Web del autor del proyecto

Los últimos avances en domótica, Vesta y Lares aparecerán en:

https://montefaro.eu/domotics

Este trabajo de fin de grado se terminó de componer el día 1 de Julio de 2022.