SEP
SECRETARÍA DE EDUCACIÓN PÚBLICA

Subsecretaría de Educación Superior
Tecnológico Nacional de México
Instituto Tecnológico de Orizaba

DIVISIÓN DE ESTUDIOS DE POSGRADO E INVESTIGACIÓN

OPCIÓN I.- TESIS

TRABAJO PROFESIONAL

"DESARROLLO DE UNA LPS PARA DOMÓTICA UTILIZANDO PROGRAMACIÓN ORIENTADA A DELTAS".

QUE PARA OBTENER EL GRADO DE:

MAESTRO EN SISTEMAS COMPUTACIONALES

PRESENTA:

Lic. Víctor Jassiel Díaz Velásquez

DIRECTOR DE TESIS:

Dr. Ulises Juárez Martínez

CODIRECTOR DE TESIS:

Dra. María Karen Cortés Verdin

ORIZABA, VERACRUZ, MÉXICO. **ABRIL 2016**

"2015, Año del Generalísimo José María Morelos y Pavón"

FECHA: 06/04/2016
DEPENDENCIA: POSGRADO
ASUNTO: Autorización de Impresión
OPCIÓN: I

C. **VICTOR JASSIEL DIAZ VELASQUEZ**
CANDIDATO A GRADO DE MAESTRO EN:
SISTEMAS COMPUTACIONALES

De acuerdo con el Reglamento de Titulación vigente de los Centros de Enseñanza Técnica Superior, dependiente de la Dirección General de Institutos Tecnológicos de la Secretaría de Educación Pública y habiendo cumplido con todas las indicaciones que la Comisión Revisora le hizo respecto a su Trabajo Profesional titulado:

"DESARROLLO DE UNA LPS PARA DOMÓTICA UTILIZANDO PROGRAMACIÓN ORIENTADA A DELTAS".

Comunico a Usted que este Departamento concede su autorización para que proceda a la impresión del mismo.

A T E N T A M E N T E

M.C. MA. ELENA GARCÍA REYES
JEFE DE LA DIV. DE ESTUDIOS DE POSGRADO

SECRETARIA DE EDUCACION PÚLICA
INSTITUTO TENOLOGICO DE ORIZABA

ggc

"2015, Año del Generalísimo José María Morelos y Pavón"

FECHA : 03/02/2016

ASUNTO: **Revisión de Trabajo Escrito**

C. M.C. MA. ELENA GARCÍA REYES
JEFE DE LA DIVISION DE ESTUDIOS
DE POSGRADO E INVESTIGACION.
P R E S E N T E

Los que suscriben, miembros del jurado, han realizado la revisión de la Tesis del (la) C. :

VICTOR JASSIEL DIAZ VELASQUEZ

la cual lleva el título de:

"DESARROLLO DE UNA LPS PARA DOMÓTICA UTILIZANDO PROGRAMACIÓN ORIENTADA A DELTAS".

Y concluyen que se acepta.

A T E N T A M E N T E

PRESIDENTE : **DR.. ULISES JUAREZ MARTINEZ** FIRMA

SECRETARIO : **DRA. MARIA KAREN CORTES VERDIN** FIRMA

VOCAL : **MTRO. LUIS ANGEL REYES HERNANDEZ** FIRMA

VOCAL SUP. : **MTRA. BEATRIZ ALEJANDRA OLIVARES ZEPAHUA** FIRMA

EGRESADO(A) DE LA MAESTRIA EN **SISTEMAS COMPUTACIONALES**

OPCION: **I** **Tesis**

Agradecimientos

Gracias a mi familia (mamá, papá, hermano) por todo el apoyo, que a pesar de la distancia siempre están pendientes de mi y nunca me abandonan.

Gracias abuelos por soportarme tanto tiempo, brindarme el calor de un hogar y darme lo que puedan a pesar de las carencias.

Gracias Fabi por siempre estar a mi lado, darme tu amistad, cariño y alegría, apoyarme en lo que puedas y ayudarme a mejorar mis hábitos.

Gracias Ana, Nely, Karen, Nancy, Ale, Moi y Job por brindarme su amistad y compañía durante estos casi tres años, siempre nos diferenciamos de las demás generaciones por estar unidos.

Gracias Doctor Ulises ya que la idea del tema fue originalmente de usted y sin su guía no hubiera logrado los objetivos del proyecto.

Gracias a mi jurado evaluador (maestro Luis, maestra Zepahua, doctora Karen) por sus comentarios, correcciones y sugerencias.

Gracias Dios por darme a la familia y amistades que tengo, gracias por siempre colocarme en el momento preciso y con la gente adecuada.

¡Gracias a todos!

Índice general

Resumen x

Abstract xi

Introducción xii

1 Antecedentes 1

1.1. Marco teórico 1

1.1.1. Domótica 1

1.1.2. Líneas de productos de software 2

1.1.3. Personalización en masa 3

1.1.4. La plataforma 4

1.1.5. Variabilidad 4

1.1.6. Marco de trabajo para la ingeniería de LPS 6

1.1.7. Ingeniería del dominio 6

1.1.8. Ingeniería de aplicación 6

1.1.9. Programación orientada a deltas 7

1.2. Planteamiento del problema 8

1.3. Objetivo general y objetivos específicos 8

1.3.1. Objetivo general 8

1.3.2. Objetivos específicos 8
1.4. Justificación 9

2 Estado de la práctica **10**
2.1. Trabajos relacionados 10
2.2. Análisis comparativo 20
2.3. Solución propuesta 24
2.3.1. Justificación 24
2.3.2. Metodología de desarrollo 25

3 Aplicación de la metodología **27**
3.1. Ingeniería del dominio 28
3.1.1. Gestión de producto 29
3.1.2. Ingeniería de requerimientos de dominio 32
3.1.3. Diseño de dominio 34
3.1.4. Implementación del dominio 41
3.2. Ingeniería de aplicación 43
3.2.1. Ingeniería de requerimientos de aplicación 43
3.2.2. Diseño de la aplicación 45
3.2.3. Implementación de la aplicación 45

4 Resultados **51**
4.1. Productos de domótica generados 51
4.1.1. Aplicación de escritorio 59
4.1.2. Aplicación móvil 63
4.2. Configurador reutilizable de LPS 66
4.2.1. Interfaz del configurador 67
4.2.2. Integración con DeltaJ 82
4.3. Controladores de ambiente domótico desarrollados 83
4.3.1. Domopi 84

4.3.2. Nodepi 87
4.4. Interoperabilidad con otros dispositivos de hardware 93
4.4.1. Arduino Rodas 94
4.4.2. Luces MiLight 95

5 Conclusiones y recomendaciones **103**
5.1. Conclusiones 103
5.2. Recomendaciones 104

Productos académicos **106**

Anexos **107**
A. Encuesta de definición de productos 107
B. Documentación de ingeniería de dominio 110
C. Documentación del producto Completo 126

Bibliografía **136**

Índice de figuras

3.1. Marco de trabajo para LPS 28
3.2. Diagrama CVL de variabilidad externa 33
3.3. Diagrama CVL de variabilidad interna 34
3.4. Diagrama de despliegue general del sistema domótico 35
3.5. Capas del sistema domótico 36
3.6. Diagrama de clases del paquete *modelos* 36
3.7. Diagrama de clases del paquete *server* 38
3.8. Bosquejo pantalla *inicio de sesión* 41
3.9. Bosquejo pantalla de bienvenida 42
3.10. Bosquejo pantalla *administración de usuarios* 42
3.11. Clases de la característica *Luces* 45

4.1. Árbol de características CVL del producto *Rodas* 52
4.2. Arquitectura del producto *Rodas* 53
4.3. Árbol de características CVL del producto *Domopi* 53
4.4. Arquitectura del producto *Domopi* 54
4.5. Árbol de características CVL del producto *Nodepi* 55
4.6. Arquitectura del producto *Nodepi* 56
4.7. Árbol de características CVL del producto *Completo* 56
4.8. Arquitectura del producto *Completo* 57

4.9. Pantalla de inicio de sesión de aplicación de escritorio 59
4.10. Pantalla de bienvenida de la aplicación de escritorio 60
4.11. Pestaña de áreas de la aplicación de escritorio 61
4.12. Pestaña de dispositivos de la aplicación de escritorio 62
4.13. Controles avanzados en la aplicación de escritorio 62
4.14. Pantallas de la aplicación móvil 64
4.15. Pantallas de la aplicación móvil (parte 2) 64
4.16. Pantalla principal del configurador 68
4.17. Pestaña de información de la LPS 69
4.18. Pestaña de características de la LPS 69
4.19. Pestaña de productos de la LPS 70
4.20. Pestaña de requerimientos de la LPS 71
4.21. Pestaña de diseño de la LPS 72
4.22. Ejemplo de combinación de diagramas de clases 73
4.23. Pestaña de implementación de la LPS 74
4.24. Pestaña de pruebas de la LPS 75
4.25. Pestaña de exportación 75
4.26. Ventana de estado de exportación 76
4.27. Exportador de DeltaJ 83
4.28. Eligiendo un producto a exportar en el configurador 84
4.29. Sensores empotrados y conectados a la Raspberry Pi 85
4.30. Diagrama de conexión de sensores a la Raspberry Pi 85
4.31. *Charlando* con la Raspberry Pi 86
4.32. Dispositivo de lectura de diversos sensores 89
4.33. Diagrama de conexión con sensores 90
4.34. Dispositivo de control de cortinas 91
4.35. Diagrama de conexión del dispositivo de cortina 91
4.36. Dispositivo de luces regulables 93
4.37. Simulador del dispositivo Arduino 95

4.38. Luz RGBW y puente WiFi de MiLight 96

Índice de tablas

2.1. Comparativa de trabajos relacionados 20
2.1. Comparativa de trabajos relacionados (continúa) 21
2.1. Comparativa de trabajos relacionados (continúa) 22
2.1. Comparativa de trabajos relacionados (continúa) 23

4.1. Características y productos de la LPS 58
4.2. Comandos del dispositivo Arduino 95

Índice de listados

3.1. Respuesta en primera versión del protocolo 39
3.2. Cambiar un estado en primera versión del protocolo 39
3.3. Información de dispositivos en segunda versión del protocolo 40
3.4. Encendiendo luces en segunda versión del protocolo 41
3.5. Módulo delta dBase 44
3.6. Módulo delta dLuces 47
3.7. Módulo delta dDomopi 49
3.8. Módulo delta dDomopiLuces 50
4.1. Ejecutando una función en el servidor 63
4.2. Ejemplo de implementación de una prueba 77
4.3. Versión muy reducida de un ejemplo del XML de un proyecto del configurador 80
4.4. Archivo de configuración de DeltaJ 81
4.5. Archivo de configuración para LIRC 88
4.6. Código fuente del dispositivo detector de gases 98
4.7. Continuación de código fuente del dispositivo detector de gases 99
4.8. Fragmento del código fuente del dispositivo de control de cortinas 100
4.9. Continuación de código fuente del dispositivo de control de cortinas 101
4.10. Código fuente del dispositivo de luces regulables 102

Resumen

Cada vez es necesario contar con mejores procesos y métodos de ingeniería que permitan a una organización sobresalir y mejorar su competitividad. Existen muchos métodos de desarrollo de software, es muy importante elegir el adecuado según el proyecto y experiencia de la organización.

En el presente documento se presenta el trabajo realizado para el desarrollo de una línea de productos de software para el dominio de domótica y la utilización del enfoque de desarrollo composicional de la programación orientada a deltas.

Al final, se obtuvo una familia de productos de software para la automatización de casas, aportando mayor seguridad, comodidad y reducción de costos para los habitantes del hogar. Se examinaron las ventajas y desventajas que implica la ingeniería de Líneas de Productos de Software y cómo se aplica con la ayuda de un lenguaje hecho especialmente para el desarrollo de LPS.

Se experimentó con plataformas de desarrollo de hardware abiertas para implementar dispositivos para el área de domótica y que sirvan de prototipo para un caso de estudio.

Se desarrolló una herramienta capaz de automatizar la generación de documentación y que trabaja de manera similar al método composicional utilizado por DeltaJ, de esta forma se logró automatizar el ciclo de desarrollo completo de una LPS.

Se utilizaron insumos de la empresa Rodas Computación S.A. de C.V. como documentos de requerimientos y prototipos de hardware para conocer las necesidades reales de los clientes en un ambiente hotelero y tomarlo como referencia para el sistema domótico.

Abstract

An organization ever needs better engineering processes and methods that allow to excel and improve their competitiveness. There are many methods of software development, is very important to choose the right one according to the project and experience of the organization.

In this document we show the working performed for development a software product line targeting domotics enviroment and the use of the compositional delta oriented programming approach.

At the end, we got a software product family for the home automation, providing safety, confort and lower costs for the home residents. We explored advantages and disadvantages implied by software product line enginnering and how it is applied within a programming language especially made to develop SPL.

Open-hardware develop platforms was tried to make devices for home automation that even helped with a study case.

A tool able to automate documentation generation was developed, use a similar compositional method like used by DeltaJ, this way framework cycle was completly automated.

Rodas Computación S.A. de C.V. artifacts was used, like requirements documents and hardware prototypes, all this to know the real customer's needs, taking hotel domain as reference for the domotic system.

Introducción

La Ingeniería de Líneas de Productos de Software (LPS) surgió al notar la manera en que se fabrican los productos que se consumen día a día. Su objetivo principal es transferir las ventajas del concepto de líneas de producción al software, es decir, mejorar el ritmo de lanzamiento de productos al mercado, lograr la personalización en masa, mejorar la competitividad de la organización y obtener mayores ingresos. A pesar de las ventajas que supone, en México no se ha reportado el desarrollo de líneas de productos en la industria de software.

Recientemente surgió el paradigma de programación orientada a deltas, cuya promesa es flexibilizar el desarrollo de Líneas de Productos de Software basándose en las diferencias que definen a un producto u otro. DeltaJ es el lenguaje que trata de probar el concepto de programación orientada a deltas, organiza el código fuente en módulos deltas con instrucciones puntuales de dónde y cómo insertar, modificar o eliminar el código para crear diferentes especializaciones de una arquitectura de referencia.

Con base a esto, el objetivo principal de este trabajo de tesis es desarrollar una línea de productos de software para el campo de domótica (automatización de hogares) aplicando el paradigma de programación orientada a deltas. Esto con el apoyo de un marco de trabajo con las mejores técnicas recomendadas para el desarrollo de LPS, el lenguaje de común de variabilidad para expresar de manera gráfica los deltas a aplicar, el lenguaje de DeltaJ para facilitar la definición/aplicación de deltas, y por último, el apoyo de una empresa que ayude a comprender el segmento de mercado de domótica y las necesidades de los usuario. Adicionalmente, se busca introducir y apoyar a adoptar todos estos conceptos a desarrolladores y empresas de la región

interesadas en mejorar sus procesos y su competitividad.

En el capítulo uno se presentan los conceptos necesarios para clarificar el alcance del proyecto; en el capítulo dos se muestra una serie artículos de investigación que se analizaron y dieron pauta al camino elegido para este proyecto, conformando así el estado del arte y solución propuesta; en el capítulo tres se documentan los pasos que se siguieron y los insumos generados de acuerdo al marco de trabajo utilizado; en el capítulo cuatro se muestran todos los productos, aplicaciones y frutos obtenidos durante el desarrollo de la LPS; y en el capítulo 5 se detallan las conclusiones y trabajo a futuro para este proyecto.

Capítulo 1

Antecedentes

Este capítulo presenta la información de referencia del proyecto de tesis, además del problema a abordar, los objetivos cumplidos y las razones que justificaron el hecho de dar solución al problema.

1.1. Marco teórico

A continuación se explican definiciones y teorías básicas para comprender por separado las diferentes conceptos que integraron y dieron forma a esta propuesta de trabajo de tesis. Se comienza de lo general hasta terminar con los conceptos más puntuales.

1.1.1. Domótica

La domótica es el campo donde la casa (*domus*) se mezcla con la tecnología en distintas formas (informática, así como mecánica, ergonomía y comunicación) [1]. Hoy en día muchos de los dispositivos en el hogar se controlan por microprocesadores. La domótica integra dichos dispositivos en una red común y de manera inteligente los hace cooperar entre sí para tomar decisiones, controlar su comportamiento y realizar tareas complejas con poca o nula intervención humana.

Los principales objetivos de la domótica son: comodidad, seguridad, cuidados de salud y reducción de costos. La comodidad se incrementa cuando tareas tediosas se realizan de manera

automática o manual desde una interfaz intuitiva y bien diseñada. La seguridad se mejora cuando se cuenta con cámaras de vídeo, alarmas y otras herramientas que permitan tomar acciones rápidas a escenarios de peligro como robos, accesos no permitidos o emergencias. De manera parecida, prevenir otros peligros para la salud humana con detectores de humo, incendios o cortos circuitos. Por último la reducción de costos se logra con administración de energía, como reguladores de calefacción o enfriamiento, luces automáticas y cualquier otra actividad que evite el consumo innecesario de energía.

Los dispositivos de una casa inteligente se dividen principalmente en dos: sensores y actuadores. Estos dispositivos son componentes electrónicos o mecánicos que miden o influencian valores físicos de su ambiente. Los sensores recogen información de su ambiente y la hacen disponible para el sistema (como un sensor infrarrojo que detecta la presencia de una persona en un cuarto). Los actuadores permiten al sistema crear cambios en un ambiente (como variaciones en la iluminación o tonalidad).

También existen los dispositivos de control inteligentes, que obtienen datos de sensores, procesan la información y activan a los actuadores si es necesario. Un dispositivo de control es responsable solo de una fracción del total de funcionalidades de una casa inteligente (parecido a un subsistema). Aunque este dispositivo actúe de manera autónoma, es necesario que publique la información que obtenga y las decisiones que tome al sistema central.

1.1.2. Líneas de productos de software

La ingeniería de Líneas de Productos de Software nació al ver el cambio significante en que se fabrican los productos consumidos día a día [2]. Al comienzo de la industrialización, los consumidores eran selectos (pocos) y los productos caros, sin embargo la industrialización fue creciendo y aplicándose cada vez a más áreas, se abarataron los insumos, el mercado creció y el número de consumidores se disparó. Un ejemplo de esto ocurrió en la industria automotriz, no se lograban producir vehículos con el ritmo deseado para satisfacer a todos los clientes. La empresa Ford fue pionera al cambiar de la manera tradicional de producción (los talleres fabricaban vehículos uno a uno en un mismo lugar) al concepto de líneas de producción, donde los vehículos

ahora se fabricaban de varios al mismo tiempo pasando por una "línea"donde paso por paso se arma el vehículo. En un paso se agregaba el chasis, en otro las llantas, en otro el motor y así componente por componente hasta terminar el automóvil. Esto mejoró los tiempos de producción, facilitó la heurística y mejoró la calidad al especializar un grupo de personas en un paso particular. También permitió la introducción de la personalización en masa, al bifurcar la línea de productos y agregar componentes diferentes (por ejemplo, colocar una caja de velocidades automática en lugar de manual, o un capote con quemacocos en lugar del tradicional) se obtiene un producto diferente y se abarcan más tipos de consumidores con gustos diferentes.

La ingeniería de productos de software se basa en este concepto, contar con un proceso de construcción de software donde uno elige que componentes incluir, el tipo de interfaz de usuario o funcionalidades extra a incluir para obtener de manera rápida una familia de productos de software y satisfacer los gustos de la mayor parte posible de clientes.

Una línea de productos de software (LPS) es un conjunto de sistemas de software con lo común y variable bien definido. En la ingeniería de sistemas individuales, los productos son independientes y auto contenidos, esto significa diferentes proyectos para desarrollar diferentes productos. En la ingeniería de líneas de productos, se desarrollan dos o más productos al mismo tiempo (incluso los que se desarrollarán en el futuro) ya sea para cubrir diferentes preferencias de los clientes finales o porque la empresa se prepara para nuevas tecnologías que surjan en el futuro y se desea sean incluidas en los productos a desarrollar.

1.1.3. Personalización en masa

A inicios de la industrialización, los productos estandarizados fueron suficientes para los clientes por algún tiempo, sin embargo, las preferencias de los clientes comenzaron a emerger y los clientes demandaron a los fabricantes más opciones personalizadas para elegir. Ese fue el comienzo de la personalización en masa, que es la producción a gran escala de productos adaptados a las necesidades individuales de clientes.

1.1.4. La plataforma

Una plataforma es la tecnología base en la cual se construye o edifican otras tecnologías o procesos. Como analogía se consideran los cimientos de una casa, esa es la base sobre la que se construye el edificio y es posible hacer el mismo tipo de cimientos para muchas casas aunque tengan diferente número de cuartos o estilo arquitectónico. La plataforma define lo que se podrá construir sobre ella y está preparada para ello. No son los mismos cimientos para una casa que para un edificio de 20 pisos. La plataforma de software se prepara para soportar todos los componentes que se necesiten para crear el producto final, y está presente en todos los productos de una misma línea de producción. La plataforma de software es el conjunto de subsistemas de software e interfaces que forman una estructura común de la cual se derivan una serie de productos desarrollados eficientemente.

1.1.5. Variabilidad

La variabilidad de una línea de productos de software es el modelado de la variabilidad que se tomará en cuenta para el desarrollo de las diferentes aplicaciones personalizadas usando artefactos predefinidos y ajustables. Por lo tanto, la variabilidad de una línea de productos también define las diferentes aplicaciones a desarrollar o lanzar para ser producidas en la línea de productos.

La variabilidad nace durante el subproceso de gestión del producto cuando se identifican las funcionalidades comunes y variables de una línea de productos de software. La variabilidad afecta a los artefactos, requerimientos, arquitectura, componentes y pruebas de los productos de software a generar. Durante la fase de requerimientos la variabilidad se maneja en un alto nivel de abstracción y se refina al avanzar en las siguientes etapas de desarrollo.

Para administrar la variabilidad y evitar se salga de control es necesario realizar los siguientes subprocesos: apoyar a las actividades interesadas en definir la variabilidad, administrar los artefactos variables, apoyar las actividades interesadas en resolver y ajustar la variabilidad y por último coleccionar, almacenar y administrar información de seguimiento necesaria que permita realizar tales tareas.

Sujeto Variable: Un sujeto variable es un elemento variable del mundo real o una propiedad

variable de tal elemento. Se identificar con la pregunta ¿qué varía? También es importante identificar el por qué varía y se debe a las necesidades del cliente, leyes del lugar, razones técnicas, entre otras razones.

Objeto Variable: Un objeto variable es una instancia particular de un sujeto variable, se identifica haciendo la pregunta *¿Cómo varía el sujeto?* y tiene que ver con las diferentes formas que un sujeto variable posiblemente toma.

Punto de Variación: Un punto de variación es una representación del sujeto variable dentro de un dominio o artefactos de dominio y se enriquece por información contextual como los detalles de la incrustación del sujeto variable en la línea de productos de software así como la razón por la que el punto de variación se introdujo.

Variantes: Una variante es una representación de un objeto variable dentro de los artefactos del dominio. Es lo que el objeto variable es al sujeto variable. Una variante identifica una opción particular de un punto de variación y se asocia con otros artefactos para indicar que dichos artefactos corresponden a una opción particular.

Variabilidad Externa: La variabilidad externa es la variabilidad de los artefactos del dominio que es visible para los clientes. Surge de las diferentes necesidades de los diferentes tipos de clientes, es responsable directamente de la satisfacción del cliente. Incluso se da por diferencias en las leyes que aplican a diferentes dominios (dominio médico o dominio de automatización de casas). También la adecuación a estándares de calidad causa variabilidad externa.

Variabilidad Interna: La variabilidad interna es la variabilidad de los artefactos del dominio que está oculta para los clientes. Surge en parte por la variabilidad externa, las diferentes decisiones que toma un cliente tienen impacto en los artefactos del dominio, arquitectura, y características de la aplicación con las que lidiarán los proveedores de la línea de productos de software. De manera similar, el refinamiento de la variabilidad interna conduce a más variabilidad interna en un nivel más bajo de abstracción. Finalmente, están las cuestiones técnicas que no tienen que ser consideradas por el cliente, por ejemplo razones de pruebas, implementación, instalación, mantenimiento, o en materia de escalabilidad, portabilidad, etcétera.

1.1.6. Marco de trabajo para la ingeniería de LPS

Para facilitar el desarrollo de la Línea de Productos de Software se hace uso de el marco de trabajo propuesto en [2] en el cual se sugiere el uso de plataformas, personalización en masa (tomando en cuenta la variabilidad), reutilización de artefactos de diseño e implementación. Este marco de trabajo (figura 3.1) se explica con más detalle en el capítulo tres.

1.1.7. Ingeniería del dominio

Es la primera etapa definida en el marco de trabajo propuesto en [2] para la realización de una línea de productos de software. En esta etapa se define lo común y lo variable de una línea de productos de software así como las aplicaciones de software planeadas y el alcance de ellas. Se definen y construyen artefactos reutilizables que cumplan con la variabilidad deseada. Consta de cinco subprocesos en cada uno de los cuales se detalla y refina la variabilidad definida en el subproceso anterior, también se retroalimenta la factibilidad de realizar la variabilidad planeada. El principal objetivo de esta etapa es el desarrollo de la plataforma de software.

1.1.8. Ingeniería de aplicación

La Ingeniería de Aplicación es la segunda etapa definida en el marco de trabajo propuesto en [2] para la implementación de una línea de productos de software. En esta etapa se explota las posibilidades de la plataforma y la variabilidad definidas en la ingeniería de dominio, se reutilizan los insumos de dominio tanto como sea posible, se documentan los artefactos de la aplicación (como requerimientos, arquitectura, componentes y pruebas) y se relacionan con los artefactos elaborados en la ingeniería de dominio, se enlaza la variabilidad con los requerimientos de una aplicación para extender la arquitectura, componentes y casos de uso, por último se estima el impacto de las diferencias de los requerimientos de dominio y aplicación en las arquitecturas, componentes y pruebas. El principal objetivo de esta etapa es el desarrollo de un producto específico planeado en la línea de productos de software.

1.1.9. Programación orientada a deltas

La programación Orientada a Deltas (propuesta en [3]) es un enfoque modular para la implementación de una línea de productos de software. Un delta es la variación de una variable o función. Un módulo delta es una generalización de una característica que permite la agregación, eliminación o modificación de una clase, método o campo y aplican para una o más características. La plataforma de una línea de productos se implementa en los módulos núcleo, desde este núcleo todos los productos de la línea de productos se generan a partir de los módulos delta. Incluso es posible tomar en cuenta un producto terminado como núcleo y aplicar módulos delta para crear un nuevo producto a partir de ese.

DeltaJ: Surgió en [3] como caso de estudio para la programación orientada a deltas. Es un lenguaje basado en Java que agrega una capa de abstracción y permite organizar las clases en módulos. Hay dos clases de módulos: núcleo y delta. Un módulo núcleo es una simple colección de clases mientras un módulo delta es un conjunto de operaciones que permiten agregar, eliminar o modificar clases declaradas en otros módulos núcleo o delta.

El lenguaje trabaja a nivel de código fuente y permite definir las funcionalidades a implementar en la línea de productos de software, las combinaciones de estas funcionalidades que definen un producto de software y los módulos deltas que contienen las condiciones y modificaciones a realizar para cada producto.

Javassist: Es un marco de trabajo que facilita la manipulación de bytecode en Java. Permite a las aplicaciones Java definir nuevas clases en tiempo de ejecución o modificar archivos de clases antes de cargarlos a la máquina virtual. Javassist trabaja a nivel de código fuente o a nivel de bytecode, es capaz de editar una clase sin conocer las especificaciones del archivo compilado, insertar bytecode en forma de código fuente y compilarlo en tiempo de ejecución, asistir en la programación orientada a aspectos agregando nuevos métodos a una clase y cortes antes, después o durante la llamada o ejecución de métodos. Javassist también ofrece un editor que permite a los desarrolladores editar directamente una clase compilada.

1.2. Planteamiento del problema

La industrialización del software exige cambiar el desarrollo a la medida de un producto único por el desarrollo de una familia de productos enfocado en un mercado o segmento de mercado. El desarrollo de Líneas de Productos de Software (LPS) ha integrado de manera exitosa paradigmas de desarrollo, patrones, modelos de rasgos, marcos de trabajo, lenguajes y diversas formas de automatización.

La programación orientada a deltas (DOP - *Delta-Oriented Programming*) [3] ha mostrado su utilidad en la implementación de LPS al flexibilizar el desarrollo orientado a características. Una delta es un módulo con capacidades de agregar, modificar o eliminar código de una clase. Similar en sus capacidades a la orientación a aspectos y herramientas como Javassist, las deltas son un candidato adecuado para soportar los artefactos de una LPS, así como en la generación de código de nivel industrial.

La domótica es un área de automatización de interiores dedicada a la parte de casas habitación. La problemática se centra en cómo generar una LPS para domótica aplicando la programación orientada a deltas. La LPS se desarrollará con base en requerimientos reales de la empresa RODAS Computación S.A. de C.V.

1.3. Objetivo general y objetivos específicos

Los siguientes objetivos definen el alcance del presente proyecto de tesis:

1.3.1. Objetivo general

Desarrollar una línea de productos de software para el dominio de domótica aplicando el enfoque composicional de DeltaJ y utilizando insumos de la empresa RODAS Computación S.A. de C.V.

1.3.2. Objetivos específicos

1. Definir el segmento de mercado de domótica.

2. Revisar, adecuar y aplicar el marco de trabajo para el desarrollo de Líneas de Productos de Software integrando módulos deltas.

3. Revisar el Lenguaje Común de Variabilidad (CVL, *Common Variability Language*) para el trabajo con deltas.

4. Revisar y aplicar DeltaJ para la implementación de la LPS.

5. Utilizar insumos proporcionados por la empresa RODAS Computación S.A. de C.V.

1.4. Justificación

El desarrollo de LPS reales en México es prácticamente nulo. Con este proyecto de tesis se pretende potenciar el desarrollo de familias de productos utilizando las tendencias más actuales que van más allá de los rasgos (*features*) y extender el conocimiento a desarrolladores y organizaciones interesadas en mejorar sus procesos u ofrecer una gama de productos para un dominio específico.

Utilizar el enfoque de desarrollo de la programación orientada a deltas representa una oportunidad de innovación que abre nuevas líneas de investigación y proyectos a futuro. Los deltas representan diferencias que existen entre diferentes implementaciones de productos, prometen flexibilizar el desarrollo de Líneas de Productos de Software y facilitar la implementación.

La domótica es un área de la tecnología que pretende automatizar las tareas del hogar, no hay muchas empresas dedicadas a este rubro y el software disponible carece de integración entre distintos proveedores, esto representa una oportunidad de mercado que permite establecer relaciones con empresas de la industria de software y oportunidades de negocio.

Capítulo 2

Estado de la práctica

En este capítulo se presentan las referencias y análisis de trabajos relacionados con el proyecto de innovación, para así obtener información relevante y adquirir conocimiento acerca de diferentes enfoques desarrollados en el pasado.

2.1. Trabajos relacionados

Para sustentar este proyecto de tesis, se consultaron diversos artículos que se relacionan con la temática del proyecto presente.

A continuación, se presenta una serie de diversos artículos consultados cuya temática está relacionada con la del proyecto actual de tesis.

Una línea de productos de software (LPS) es un conjunto de sistemas de software con lo común y la variabilidad bien definida. La variabilidad se expresa en las diferentes características de un producto. En [3] se propuso la Programación Orientada a Deltas (*Delta-Oriented Programming*, DOP) como un nuevo enfoque para los lenguajes de programación y diseñada particularmente para la implementación de Líneas de Productos de Software basándose en el concepto de deltas de programas. Su objetivo es reducir las desventajas y restricciones de la Programación Orientada a Características (*Feature-Oriented Programming*, FOP) y proveer un lenguaje expresivo y flexible para las Líneas de Productos de Software. En DOP la implementación de una LPS está dividida en un módulo núcleo y un conjunto de módulos delta. El módulo núcleo es un conjunto de clases

que implementan un producto completo con una configuración completa, es decir, tomar como base un producto completo y obtener desde él otros productos nuevos completos, esto permite aplicar las mismas técnicas de desarrollo que se usan en la ingeniería de aplicaciones únicas para asegurar su calidad. Los módulos delta especifican cambios que se deben hacer en el núcleo para generar nuevos productos, así, se modifican las clases cambiando su superclase o su constructor, o agregando/eliminando métodos y campos. Un módulo delta contiene condiciones que determinan para qué característica específica se aplican las modificaciones programadas. La idea general de la Programación Orientada a Deltas no está restringida a un solo lenguaje de programación, para demostrar su factibilidad se implementó en Java introduciendo el lenguaje de programación DeltaJ. Usando un sistema de tipos basados en restricciones, se asegura que la implementación de la LPS está bien construida y libre de conflictos o ambigüedades. Para evaluar el potencial de DOP se comparó con FOP, conceptual y empíricamente.

En [4] se presenta un lenguaje de Programación Orientada a Deltas para Java 1.5 extendiendo las conceptos realizados y probados en [3]. Esta herramienta prototipo, llamada DeltaJ 1.5, provee un acceso total e integrado a las funciones de Java. A diferencia de su predecesor, esta extensión incluye operaciones delta para integrarse totalmente con el sistema de paquetes de Java, declarar y modificar interfaces, cambiar explícitamente la jerarquía de herencia, acceder a tipos anidados y enumerados, alterar definiciones de campos, y (sin ambigüedad) eliminar métodos sobrecargados. Además, se mejoró la especificación de la declaración de la línea de productos por medio de un lenguaje separado. Se comparó DeltaJ con Eclipse RCP (*Rich Client Platform*, un *plugin* utilizado para desarrollo de software modular y flexible, enfocado a soportar sistemas de variabilidad extensa) a través de un caso de estudio llamado *SimpleTextEditor*, esta línea de productos constó de 11 características que resultan en 128 variantes válidas. Después de desarrollar la misma línea de productos con ambas herramientas se concluyó que DeltaJ es suficientemente expresiva para implementar un sistema variante de la vida real; de igual manera el código fuente no es redundante y menos ambiguo que con Eclipse RCP. Aunque DeltaJ 1.5 integra la mayoría de las características del lenguaje Java, aún se planea dar soporte de operaciones delta para manipular anotaciones, firmas de método (incluyendo modificadores), expresiones lambda, métodos anónimos, declaraciones *switch-case* y bloques *multi-catch*.

En cualquier casa común se realizan actividades que requieren de una gran variedad de tareas repetitivas que trabajan aisladas unas de otras. Es así como el hogar (conjunto de actividades) tiene un gran potencial de convertirse en un sistema distribuido de servidores colaborativos. La domótica es el campo donde la casa (*domus*) se mezcla con la tecnología en distintas formas (informática, así como mecánica, ergonomía y comunicación). Los elementos de la domótica son heterogéneos en todos sus aspectos. Es por esto que en [1] se propuso el uso de servicios Web para resolver problemas de licencia, escalabilidad, heterogeneidad y topología en un sistema distribuido de hogar. Los servicios Web permiten la comunicación tipo cliente-servidor así como igual-a-igual. Se implementó el marco de trabajo propuesto llamado ITEA en un caso de estudio concreto para monitorizar el estado de salud de un adulto mayor. La casa se equipó con una red heterogénea sin cables (WIFI, Bluetooth y radio frecuencias dedicadas), una variedad de sensores fijos y portables transformados en un sistema distribuido usando servicios Web para coordinarse y monitorear el estado del habitante del hogar. Se descartó la comunicación a través del suministro eléctrico, ya que el protocolo X10 y otras evoluciones no permiten demasiados mensajes concurrentes y se limitan al control de luces y persianas. Se descartaron igual protocolos y servicios de terceros que suelen ser de licencia privada, ya que la escalabilidad solo es posible que se extienda con el mismo proveedor que desea mantener un monopolio, además de que la mayoría manejan la heterogeneidad pobremente. El paradigma fundamental de un servicio Web es publicar, descubrir e invocar. Un servicio Web publica sus funcionalidades, otros servicios Web los descubren y utilizan después. Se basa en una familia de protocolos basados en XML, un estándar totalmente público y extensible. Un servicio Web no se limita a una topología, es posible usar cliente-servidor o igual-a-igual (P2P). Los sensores y actuadores no necesitan entender XML o WS pero sus controladores sí. La propuesta de implementar una pila de servicios Web se describió en dos palabras: total interoperabilidad. Ya sea que un dispositivo implemente un servicio Web o los más sencillos se conecten a través de un controlador que implemente el servicio Web.

Los programas o productos generados en una línea de productos de software se sintetizan por módulos compuestos que implementan características particulares. Además de las restricciones de dominio de alto nivel que gobiernan la compatibilidad de las características entre sí, también

hay restricciones de bajo nivel de implementación, por ejemplo, que un módulo de características tenga referencias a clases de otro módulo. En [5] se discutió que para resolver este problema se necesita de una *Composición Segura* para garantizar que los productos compuestos de diversos módulos de características estén ausentes de referencias a clases, métodos o campos indefinidos. Por lo que en [5] se demuestra cómo garantizar la composición segura en líneas de productos AHEAD (Algebraic Hierarchical Equations for Application Design, Ecuaciones Algebraicas Jerárquicas para el Diseño de Aplicaciones) usando modelos de características y solucionadores de satisfacción. Para analizar las dependencias entre módulos de características, se necesitaron saber las relaciones entre clases, métodos y referencias variables en cada módulo, para esto se usó una técnica pionera en HiperJ: compilar un módulo manteniendo copias en el mismo módulo de clases o métodos que no están en ese mismo módulo, haciéndolos declarativamente completos. Para crear un módulo hay dos opciones, combinar y compilar el código fuente o (la recomendada) compilar los módulos de características y que AHEAD combine y genere el bytecode para producir directamente el producto terminado. La primera opción hizo que los errores ocurrieran durante la ejecución y la segunda opción informó los errores al momento de la composición. Estas técnicas ayudaron a identificar errores no conocidos en líneas de productos ya existentes, crear mejores diseños y evitar aquellos no satisfactorios. Incluso las pruebas de rendimiento con solucionadores de satisfacción fueron alentadores, se realizaron en 30 segundos en líneas de productos "pesadas".

En [6] se declaró que la generación de Líneas de Productos de Software usando características es una tarea complicada. Algunas implementaciones de Líneas de Productos de Software soportan características con una granularidad burda (por ejemplo, la habilidad de agregar o envolver métodos enteros). Sin embargo, extensiones más refinadas permiten agregar declaraciones en el medio de un método que requieren de operaciones complejas u ofuscar el código fuente con anotaciones. Aunque se han implementado ya exitosamente Líneas de Productos de Software con esta granularidad burda, se necesitan extensiones que manejen la granularidad más refinadamente cuando se manejen características en aplicaciones heredadas. Así algunas Líneas de Productos de Software se beneficiaron con reducción de código replicado o mejor legibilidad en él. En [6] se analizaron los efectos de la granularidad de características y se presentó una herramienta

llamada *Colored IDE* basada en Eclipse que permite a las características ser implementadas con mejor granularidad en una extensión más refinada. En dos casos de estudio se comparó cómo la herramienta simplificó y mejoró el desarrollo de Líneas de Productos de Software contra la manera tradicional.

En [7] se describió un método para la migración de multiples instancias de un sistema de información único a una línea de productos de software. El despliegue de una línea de productos de software es capaz de tratar con variantes que cambian con el tiempo de una manera más rentable. Se propusieron y usaron arquitecturas federadas que dividen el software en capas de tipos genéricas. Estas son el corazón de las líneas de productos ya que permiten una mejor integración empresarial y resistencia al cambio. En un caso de estudio se migró un gran sistema global de transacciones y liquidación con muchas variantes específicas de sitio. Con esto se logró que: unidades de código global solo se alteren cuando se necesite impactar en todos los sistemas, que las unidades de código locales solo tengan requerimientos específicos del sitio, además de un sistema más flexible que tiene la habilidad de crecer rápido y mayor competitividad.

Debido a la creciente complejidad del desarrollo de Líneas de Productos de Software y los diversos interesados, los requerimientos con frecuencia se especifican desde vistas parciales y heterogéneas. En [8] se presentó un lenguaje de composición de vistas múltiples para los requerimientos de Líneas de Productos de Software: el Lenguaje de Modelado de Variabilidad para los Requerimientos (VML4RE, *Variability Modeling Language for Requirements*). Este lenguaje describe cómo los elementos de los requerimientos que se expresan en diferentes modelos se componen para generar un producto específico de la línea de productos. El uso de VML4RE se ilustró en un caso de estudio de una línea de productos de software para la automatización de casas utilizando modelos de requerimientos basados en UML. El lenguaje también se validó con otros casos de estudio más pequeños. Es así como VML4RE respondió la pregunta "¿Cómo componer elementos definidos en modelos de requerimientos separados y heterogéneos usando un conjunto simple de operadores?". El lenguaje es una gran contribución al campo de lenguajes para el manejo de requerimientos en Líneas de Productos de Software debido a sus características únicas: Cada bloque de características concentra un conjunto cohesivo de acciones que se transforman en modelos de múltiples vistas para los requerimientos; tiene

nuevos operadores de composición hechos especialmente para soportar requerimientos canónicos y se basa en un vocabulario familiar para los ingenieros de requerimientos; tiene una separación explícita entre el modelado de la variabilidad y los requerimientos sin forzar la inclusión intrusiva de elementos relacionados con la variabilidad en los modelos de requerimientos; los nuevos operadores agregan, eliminan o reemplazan partes de los modelos para soportar variabilidad inclusiva o excluyente; el uso de referencias facilita la creación de composiciones y la generación de enlaces de trazabilidad. Se piensa mejorar la usabilidad del lenguaje agregando una sintaxis gráfica concreta.

Las Líneas de Productos de Software es un enfoque bien conocido para desarrollar sistemas adaptables de software para la industria. Se utilizan en dominios donde se desea software se alta calidad. En [9] se discutió que esta abrumadora diversidad de productos crea un desafío para asegurar la corrección del software generado, por lo que se propuso la división orientada a deltas, un enfoque para reducir el esfuerzo de verificación deductiva en una línea de productos donde los productos individuales son programas Java y sus relaciones se describen por deltas. Del lado de las especificaciones, se extendió el lenguaje delta para lidiar con especificaciones formales. Del lado de la verificación se combinaron *rebanadas confiables* y reutilización de *similitud guiada* confiable para facilitar el proceso de verificación. Trabajando con esta verificación de Líneas de Productos de Software, se identificaron líneas de investigación muy interesantes. La mayoría de estas verificaciones abarcan la transición que va desde un modelado sintáctico hacia un modelado más semántico de Líneas de Productos de Software. Hasta ahora, las operaciones de los módulos deltas (para código) a sus condiciones de aplicación son en su mayoría sintácticas. Se espera obtener mayor poder y precisión agregando más información semántica.

En [10] se comentó que la evolución de sistemas con programación orientada a deltas se dificulta con el tiempo debido a la agregación o cambios en los requerimientos y necesitan de mayor esfuerzo en su mantenimiento para preservar su valor. Se propuso la recomposición como una serie de transformaciones de programa que preservan su comportamiento y mejoran el diseño y estructura de sistemas de software orientados a objetos. Hay una falta de soporte para la recomposición en las Líneas de Productos de Software ya que en ellas es una tarea más compleja que en la ingeniería de aplicaciones únicas. Para hacer la recomposición en las Líneas

de Productos de Software se tiene que conservar el comportamiento de (probablemente) miles de programas en lugar de solo uno. Se abordó la recomposición de Líneas de Productos de Software presentando un catálogo de recomposición para Líneas de Productos de Software orientadas a deltas. Adicionalmente se proporcionó una guía de códigos para apoyar a los desarrolladores a identificar oportunidades potenciales de recomposición. Se demostró que la recomposición mejora el mantenimiento de una línea de productos y cómo evoluciona a través del tiempo. Debido a que la recomposición manual de líneas de productos es tediosa y no factible, se desarrolló también una implementación para aplicar la recomposición de manera semiautomática.

En [11] se menciona que los sistemas de módulos permiten una estrategia de *divide y vencerás* para el desarrollo de software. Para implementar la variabilidad en tiempo de compilación en las Líneas de Productos de Software los módulos se combinan de diferentes formas, de esta forma, la variabilidad dictamina también una descomposición dominante. Como alternativa se introdujo un sistema de módulos consientes de la variabilidad que soporta variabilidad en tiempo de compilación dentro de un módulo o sus interfaces. Así, cada módulo se considera una línea de producto cuya tipificación se comprueba aisladamente. El sistema de módulos rompe con la tradición anti-modular de un modelo de variabilidad global y provee un camino lleno de ecosistemas de software y líneas de productos de líneas de productos desarrolladas de manera abierta. Se analizó el diseño e implementación de dicho sistema de módulos en un cálculo central y suministró una implementación para el lenguaje C como parte del proyecto *TypeChef*. La implementación soportó variabilidad dentro de los módulos con directivas de pre procesamiento *#ifdef* y enlace variable a nivel de composición. Así se comprobó la tipificación de todas las configuraciones de todos los módulos de un proyecto línea de productos de software de código abierto Busybox con 811 opciones de compilación, se validaron los enlaces de todas las configuraciones y reportó la tipificación encontrada así como errores de enlace.

Según [12] en el contexto de ingeniería de aplicaciones únicas, la recomposición es comúnmente aceptada en el proceso de reestructurar el cuerpo de un código existente para mejorar su estructura interna sin afectar su comportamiento externo. Este proceso es vital para el mantenimiento y evolución de los sistemas de software. Como en la ingeniería de Líneas de Productos de Software los sistemas crecen en tamaño y complejidad, mantener una estructura limpia se vuelve

aún más importante. Se considera a la recomposición como una amplia variedad de actividades de reingeniería y se vuelve difícil ver cómo esas actividades darán persistencia a la mantenibilidad y evolución actuales y sus relaciones entre sí. Para mejorar esta situación se identificaron las dimensiones afectadas durante la reingeniería de Líneas de Productos de Software, se derivó una taxonomía que distingue y relaciona esas actividades de reingeniería, se propusieron definiciones para las tres principales bifurcaciones de la taxonomía y se clasificaron algunos trabajos existentes. Esta clasificación indica que muy pocas técnicas de implementación basadas en la composición requieren poco trabajo de recomposición de variantes persistentes.

En [13] se comentó que el desarrollo de Líneas de Productos de Software busca la reutilización modelando artefactos comunes y variables separadamente en una familia de productos. El desarrollo de software orientado a aspectos pretende separar las diferentes necesidades en *aspectos* para mejorar la modularidad, reutilización, mantenibilidad y evolución del software. Se aplicó el modelado de casos de uso orientados a aspectos al sistema de modelado actual de líneas de productos. Una especificación de casos de uso captura las necesidades del cliente como interacciones entre el sistema y los actores. En un trabajo previo se generó una herramienta llamada *Use Case Editor* que genera diagramas de casos de uso a partir de una oración escrita en lenguaje natural estricto. Se modela la variabilidad y lo común de una línea de productos de software como casos de uso y se enlazan con una relación de tipo *«variabilidad»*, la cual es una representación especializada de un aspecto. La herramienta generó así diferentes artefactos que se utilizarían durante el desarrollo de la línea de productos. En la implementación de esta herramienta existieron algunas limitaciones, por ejemplo, no es posible especificar un orden que asegure la correcta composición de los aspectos.

Para [14] las Líneas de Productos de Software son sistemas complejos y largos cuya intención es la reutilización de componentes en un conjunto de productos concretos. Las Líneas de Productos de Software definen implementaciones y arquitecturas reutilizables o comunes en todos los productos generados, y en cada producto generado hay diferencias que lo hacen único. Tales diferencias se definen como *variabilidad* y cada punto distinto como *punto de variación*. Se desarrolló un marco de trabajo conceptual para definir los conceptos de variabilidad y se discutió cómo la variabilidad se maneja en las Líneas de Productos de Software. Se creó una

notación para expresar la variabilidad en términos de características; tres patrones recurrentes de variabilidad y cómo manejar la variabilidad. Como consecuencia se incrementó el conocimiento y entendimiento del concepto de variabilidad realizando casos de estudio.

Según [15] las arquitecturas de Líneas de Productos de Software han estado bajo continua observación en la comunidad de investigación de software durante los últimos años. Aunque se han establecido diversos métodos para crear Líneas de Productos de Software no hay estudios que comparen dichos métodos. Se conocen cinco métodos que responden a las necesidades de las Líneas de Productos de Software: COPA, FAST, FORM, KOBRA y QADA. Se introdujo un marco de trabajo de evaluación para comparar los métodos de diseño antes mencionados. El marco de trabajo considera a los métodos desde el punto de vista del contexto, del usuario, de estructura y validación. Esta comparación reveló ideologías distinguibles entre los métodos, los métodos no se traslapan a pesar de estar enfocados a lo mismo (desarrollo de líneas de productos). Se concluyó que dichos métodos no parecen competir unos con otros ya que se enfocan a objetivos especiales específicos. COPA se concentra en balancear los enfoques de alto nivel y los de bajo nivel cubriendo todos los aspectos de la ingeniería de líneas de productos. FAST tiene un proceso de descripción orientado a familias con actividades, artefactos y roles, es muy adaptable pero no tan aplicable. FORM es un método orientado a características para capturar lo común dentro de un dominio, se extiende incluso para cubrir el diseño arquitectónico y desarrollo de insumos de código. KOBRA es un método práctico y simple basado en componentes usando UML, se adapta al desarrollo de sistemas únicos así como familias de productos. QADA se concentra en el diseño de arquitectura para mejorar la calidad de los requerimientos, provee soporte a la evaluación de calidad paralela de arquitecturas de Líneas de Productos de Software.

Para [16] las Líneas de Productos de Software basadas en modelos combinan la capacidad de abstracción del desarrollo de software basado en modelos y la capacidad de gestionar la variabilidad de Líneas de Productos de Software. El modelado de características es una técnica para representar la variabilidad y lo común de un conjunto de aplicaciones de una forma concisa y taxonómica, genera artefactos a nivel de requerimientos que son útiles para especificar el alcance de una línea de productos de software. Las características son requerimientos funcionalidades y no funcionales prominentes de los sistemas. Los modelos de características son jerarquías

de características que describen diferentes formas de variabilidad. Es posible que sean tanto obligatorias como opcionales. En este trabajo se presentó una forma particular para modelar las características las cuales se basan en plantillas. Como prueba del concepto se construyeron una serie de herramientas que soportan tal enfoque. El conjunto incluye una extensión para Eclipse para modelar las características y configuración, una extension que agrega soporte a las plantillas de modelos al modelador de IBM *Rational Software Modeler* y por último un verificador de plantillas que determina qué instancias se generan de configuraciones de características válidas.

Uno de los argumentos a favor del software basado en componentes es la idea de construir sistemas de software a partir de la composición de componentes ya existentes en lugar de volver a desarrollar componentes con funcionalidad idéntica o parecida. Desafortunadamente, integrar componentes existentes significa adaptarlos y usarlos en lugar de solo usarlos, lo que hace al desarrollo basado en componentes difícil de practicar. La ingeniería de Líneas de Productos de Software toca este problema creando componentes tan genéricos como sea necesario para una familia de productos particular permitiendo su reutilización. Tales componentes cubren la variabilidad y en su desarrollo e implementación. En [17] se describió un proceso para implementar componentes genéricos de líneas de productos y se proporcionó una visión general de los mecanismos de variabilidad a nivel de implementación, y se ilustró un ejemplo para realizar pruebas a componentes genéricos. El análisis resultó en una serie de patrones para implementar la variabilidad, cada patrón describe el tipo de variabilidad cubierta, contextos de aplicación útiles y sus respectivos mecanismos de implementación técnicos. Durante la implementación, tales patrones se aplican dependiendo de la variabilidad realizada. La experiencia generada por la aplicación de los patrones proporciona directamente una retroalimentación al análisis del lenguaje y posiblemente causa algún cambio en los patrones de variabilidad.

El manifiesto Ágil se conoce por ser el pilar de las metodologías Ágiles, en él se describen prácticas auto descriptivas que toda metodología ágil tiene, se dice que es ágil por reducir la cantidad de trabajo a realizar y realizar entregas tempranas al cliente y dar competitividad a la organización. Según [18] la principal ventaja de las líneas de productos es la reutilización de componentes que crea *activos* de software. Estos activos se desarrollan de manera incremental (poco a poco), de manera proactiva (se pone mayor esfuerzo a los elementos reutilizables o

del núcleo) y reactiva (nuevos productos son creados desde otros productos terminados). Las prácticas (establecidas por la SEI) que debe tener una línea de productos se dividen en tres categorías: Ingeniería, Gestión Técnica y Gestión Organizacional. Se utilizó la ingeniería de métodos para tomar partes de ambos mundos. Se tomó en cuenta dimensiones como lo común, la volatilidad del dominio y magnitud del producto. Las características en común, las técnicas ágiles y de línea de productos son colaborativas, tales colaboraciones generan un circuito continuo de retroalimentación. Los métodos ágiles aceptan los cambios y en LPS hay un alcance muy bien definido. A pesar de que no se definió un solo método para combinar las ventajas de ambos mundos, se crearon una serie de combinaciones derivadas de varios experimentos que han logrado retener en su mayoría las ventajas de cada método de desarrollo por separado y mejorar las prácticas de la organización. Se deben considerar los objetivos de las actividades a realizar para implementar métodos híbridos que sean ágiles y orientadas a la reutilización al mismo tiempo. Esos métodos híbridos lograron hacer cumplir los objetivos de la empresa y hacer contribuciones estratégicas significantes a la organización.

2.2. Análisis comparativo

A continuación en la Tabla 2.1 se presenta un cuadro comparativo de los artículos antes citados.

Tabla 2.1: Comparativa de trabajos relacionados

Autores	Título	Problema	Objetivo
Ina Schaefer, Lorenzo Bettini, Viviana Bono	Delta-oriented programming of software product lines	Poca flexibilidad en la programación Orientada a Características (FOP) para implementar LPS	Se proporcionó mayor flexibilidad implementando la Programación Orientada a Deltas e introducir el lenguaje DeltaJ

Tabla 2.1: Comparativa de trabajos relacionados (continúa)

Autores	Título	Problema	Objetivo
Marco Aiello	The Role of Web Services at Home	La poca o nula interoperabilidad entre dispositivos y servicios de diferentes proveedores impacta la escalabilidad y reutilización de los sistemas de domótica.	Se propuso la implementación de servicios Web para garantizar la interoperabilidad de todos los servicios y dispositivos.
Christian Kästner, Sven Apel, Martin Kuhlemann	Granularity in Software Product Lines	En la programación orientada a características, las características suelen implementarse con una granularidad burda y requieren de operaciones complejas u ofuscar el código para crear extensiones más refinadas.	Se propusieron herramientas como la "ColoredIDE" que permiten a las características ser implementadas con una granularidad más refinada y simplificar y reducir el código fuente.
D. Faust, C. Verhoef	Software Product Line Migration and Deployment	Es difícil para un sistema heredado el adecuarse a los nuevos requerimientos que surgen en una organización, o tratar con variantes que cambian con el tiempo.	Se describieron técnicas que ayudan a migrar un sistema heredado a una línea de productos de software para mejorar la integración empresarial del sistema y hacerlo más fuerte a los cambios.

Tabla 2.1: Comparativa de trabajos relacionados (continúa)

Autores	Título	Problema	Objetivo
Mauricio Alférez, João Santos, Ana Moreira	Multi-view composition language for software product line requirements	Debido a la complejidad del desarrollo de Líneas de Productos de Software los requerimientos con frecuencia se especifican desde herramientas de modelado con vistas parciales y heterogéneas.	Se presentó el lenguaje de composición de vistas múltiples VML4RE que describe cómo los elementos de los requerimientos expresados en diferentes modelos se deben combinar para generar un producto específico de la línea de productos de software.
Sandro Schulze, Oliver Richers, Ina Schaefer	Refactoring delta oriented software product lines	La evolución de sistemas con programación orientada a deltas se dificulta con el tiempo debido a la agregación o cambios en los requerimientos y necesitan de mantenimiento para preservar su valor	Se propuso la recomposición como una serie de transformaciones de programa que preservan su comportamiento y mejoran el diseño y estructura de sistemas de software orientados a objetos.

Tabla 2.1: Comparativa de trabajos relacionados (continúa)

Autores	Título	Problema	Objetivo
Christian Kästner, Klaus Ostermann, Sebastian Erdweg	A variability-aware module system	Para implementar la variabilidad en tiempo de compilación en las Líneas de Productos de Software los módulos se combinan de diferentes formas, sin embargo de esta forma la variabilidad dictamina también una descomposición dominante	Se propuso como alternativa un sistema de módulos consientes de la variabilidad que soportan variabilidad en tiempo de compilación dentro de sí o sus interfaces. Así, cada módulo se considera una línea de producto probada aisladamente.
Krzysztof Czarnecki, Micha Antkiewicz	Model-driven software product lines	El modelado de lo común y lo variable en una línea de productos se realiza por medio de jerarquías de características opcionales u obligatorias.	Se presentó una forma particular para modelar las características de la variabilidad basándose en plantillas y se desarrollan una serie de herramientas que dan soporte a tal enfoque.
Dirk Muthig, Thomas Patzke	Generic Implementation of Product Line Components	La reutilización de componentes de software existentes se dificulta al tener que adaptarlos en lugar de sólo usarlos.	Se describió una serie de patrones de variabilidad para la generación de componentes genéricos que se usan dentro y fuera de una familia de productos de software.

Se concluyó a partir de la tabla 2.1 que para el desarrollo de la línea de productos de software se tomen en cuenta las siguientes consideraciones: (1) utilizar la Programación Orientada a Deltas para flexibilizar el desarrollo de la línea de productos de software; (2) utilizar servicios Web en paralelo a otros protocolos de comunicación para aumentar la interoperabilidad y escalabilidad de los sistemas de domótica; (3) tomar en cuenta los patrones descritos en [17] para desarrollar componentes genéricos y maximizar su reutilización en otros proyectos; y (4) utilizar una herramienta como VML4RE para generar distintos modelos de los requerimientos y dar seguimiento de trazabilidad a los artefactos de desarrollo.

2.3. Solución propuesta

A continuación se justifica la solución propuesta para después detallar la metodología de desarrollo a seguir.

2.3.1. Justificación

Tomando en cuenta la flexibilidad que DeltaJ promete para el desarrollo de Líneas de Productos de Software, este se determinó como la mejor opción debido a que es un enfoque hecho a la medida para el problema, el desarrollo de las aplicaciones se realizará con ingeniería directa por lo que solo se trabaja con código fuente (sin tocar el *bytecode*) y la poca documentación es una oportunidad de innovación.

En [19] se menciona que los lenguajes de modelado de variabilidad CVL y OVM tienen muchas similitudes, ambos incluyen los conceptos de puntos de variación y variantes, definición de elementos obligatorios y opcionales, además de restricciones como dependencias de inclusión o exclusión, entre otros. En ambos lenguajes la variabilidad se enlaza a los modelos o artefactos base, sin embargo los lenguajes difieren en varios aspectos: (1) los modelos de variabilidad en CVL son árboles estructurados, mientras en OVM no hay una jerarquía de árbol; (2) CVL permite especificar aspectos comunes y variables de los productos de software, mientras OVM se concentra en modelar solo la variabilidad; (3) OVM diferencia entre puntos de variación y

variantes desde el nivel de especificación, mientras CVL lo hace solo al resolver la variabilidad; (4) las relaciones entre los modelos de variabilidad y los artefactos base se especifican como enlaces en OVM y como objetos en CVL; y (5) existen pequeñas diferencias en la sintaxis concreta de los lenguajes. También se menciona que CVL y OVM no difieren mucho en lo que a comprensibilidad se refiere, ambos lenguajes se recomiendan, sin embargo los usuarios (encuestados) calificaron a CLV como más comprensible que OVM por algunos pequeños inconvenientes en la notación de OVM.

Con base en esto se ha elegido a DeltaJ y CVL como la mejor alternativa para solucionar el problema de desarrollo composicional y representación de variabilidad para Líneas de Productos de Software.

2.3.2. Metodología de desarrollo

Se utilizará el marco de trabajo propuesto en [2] para la realización de la línea de productos de software. El marco de trabajo propone dos procesos principales de desarrollo: Ingeniería de Dominio e Ingeniería de Aplicación.

El proceso de Ingeniería de Dominio consta de los siguientes subprocesos:

Gestión del Producto: Su principal objetivo es lograr una mayor contribución al éxito empresarial integrando el desarrollo, producción y *marketing* de los productos que cubren las necesidades de los clientes. Controla el desarrollo, producción y *marketing* de la línea de productos de software y sus aplicaciones.

Ingeniería de requerimientos del dominio: Pretende definir los requerimientos comunes y variables del dominio y su documentación precisa. Es un proceso continuo que se refina al definir los requerimientos de todas las aplicaciones deseables en una línea de productos.

Diseño del dominio: Define los modelos o artefactos para desarrollar la arquitectura de referencia, la estructura principal del software y su textura. La arquitectura determina cómo los requerimientos, incluyendo la variabilidad, se refleja en la arquitectura.

Realización del dominio: Su objetivo es proveer un diseño detallado y la implementación de insumos de software reutilizables, basados en la arquitectura de referencia. También otros

artefactos como diseños flujos de aplicación, tablas de bases de datos, protocolos, flujos de datos, entre otros. Validación del dominio: Valida la salida de los artefactos e insumos de software de los otros subprocesos de ingeniería, principalmente los artefactos de implementación.

El proceso de Ingeniería de Aplicación consta de los siguiente subprocesos:

Ingeniería de requerimientos de aplicación: Su propósito es generar y documentar los artefactos de requerimientos específicos para una aplicación en particular y al mismo tiempo, reutilizar tanto como sea posible, los artefactos de ingeniería de dominio.

Diseño de la aplicación: Su objetivo es producir la arquitectura de la aplicación, la cual es una especialización de la arquitectura de referencia desarrollada en el diseño de dominio.

Realización de la aplicación: Pretende desarrollar a las aplicaciones específicas que se validan y prueban para que después se lancen al mercado.

Validación de la aplicación: Su meta es lograr la calidad del producto a través de pruebas de la aplicación específica y complementar las actividades de prueba en la validación de dominio.

El flujo de actividades de esta metodología es recursivo, las actividades de la ingeniería de aplicación refinan y retroalimentan a las actividades de ingeniería de dominio, la cual sirve para las aplicaciones consecuentes.

Capítulo 3

Aplicación de la metodología

En el presente capítulo se describe la metodología empleada para el desarrollo del sistema domótico aplicando el marco de trabajo propuesto en [2] y descrito en la figura 3.1 para el desarrollo de Líneas de Productos de Software. El marco de trabajo incorpora los conceptos básicos de la ingeniería de líneas de productos tradicional como el uso de plataformas o la habilidad de proveer personalización en masa.

El uso de plataformas es un método eficaz de reutilización de artefactos de software, los cuales se deben utilizar de una manera consistente y sistemática para el correcto desarrollo de las aplicaciones. Un artefacto reutilizable es cualquier tipo de artefacto de desarrollo de software como modelos de requerimientos, modelos de arquitectura, componentes de software, planes de pruebas y diseños. En la mayoría de los casos reutilizar artefactos de software sin una planeación adecuada iguala o rebasa los costos a que si se desarrollara desde cero el artefacto. Para facilitar la personalización en masa la plataforma tiene que proveer los medios necesarios para satisfacer diferentes requerimientos de diferentes clientes o interesados en el producto, aquí es donde entra el concepto de variabilidad en la plataforma, y como consecuencia, los artefactos que varían en las aplicaciones de la línea de productos son modelados utilizando variabilidad.

A lo largo de este capítulo se presentan todos los artefactos de software desarrollados para la línea de productos de software para el dominio de domótica de todas las etapas del proceso de desarrollo, de igual manera, se pretende que sirva como punto de partida para futuros proyectos

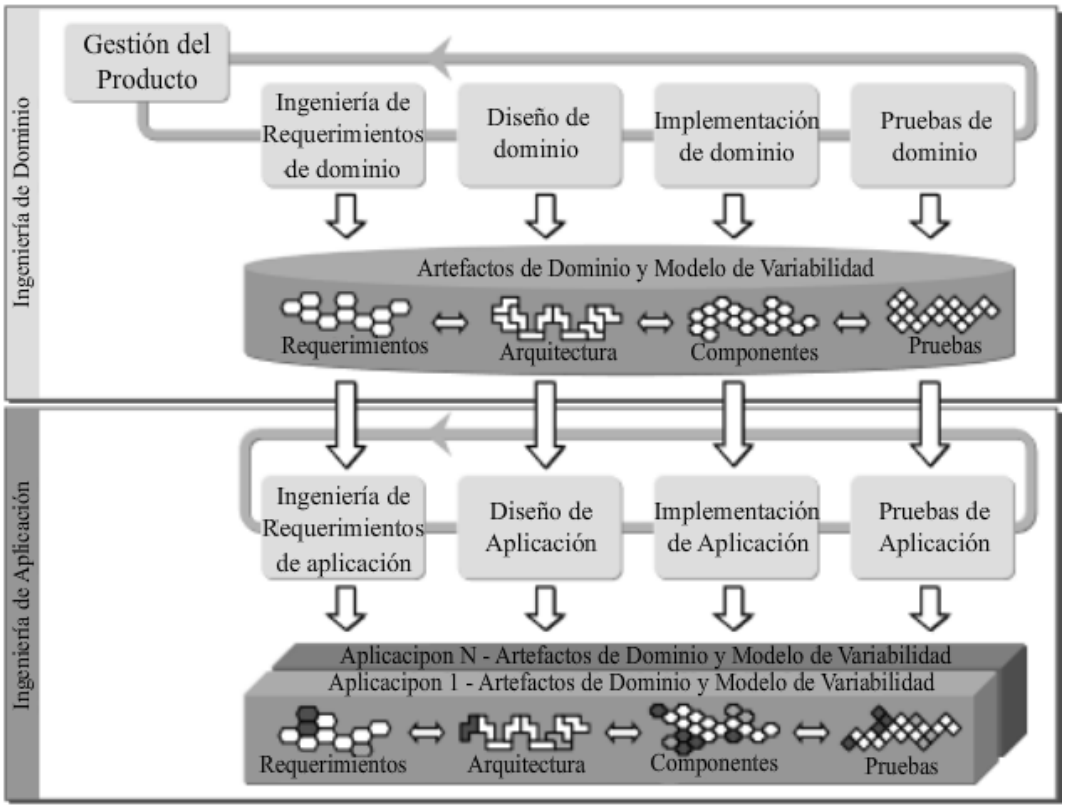

Figura 3.1: Marco de trabajo para LPS

de desarrollo de Líneas de Productos de Software para otros dominios.

3.1. Ingeniería del dominio

La ingeniería de dominio es el proceso de la ingeniería de Líneas de Productos de Software en el cual se define e implementa lo común y lo variable (variabilidad) de la línea de productos. El resultado de este proceso es una plataforma de software capaz de soportar una selección de componentes de software variables o intercambiables.

Los objetivos principales de este proceso son:

- Definir lo común y la variabilidad de la línea de productos de software.
- Definir el conjunto de aplicaciones que se planea generar.
- Establecer el alcance de la línea de productos de software.

- Definir y construir artefactos reutilizables que consigan la variabilidad deseada.

Es por esto que a lo largo de esta sección se incluye la documentación y artefactos realizados para dar vida a la plataforma de la cual se derivarán los distintos productos orientados al campo de la domótica, el proyecto se nombró *Intelidomo*.

3.1.1. Gestión de producto

El objetivo de esta etapa es contribuir al éxito de los productos generados en la línea de productos de software, abarca aspectos de desarrollo, producción y mercadeo de los productos que satisfacen las necesidades del cliente. En la etapa se tratan los aspectos económicos de la LPS y la estrategia de mercadeo. La gestión de producto es una técnica de definición de alcance para establecer lo que esta dentro y lo que esta fuera del alcance de la línea de productos. Como punto de partida se comienza definiendo los objetivos que la empresa desea cumplir con el desarrollo de la LPS, al final de la etapa se obtiene una ruta o portafolio de productos de software que la empresa desea desarrollar a corto o largo plazo. Sirve para determinar la mayor parte de características comunes y variables de presentes y futuros productos, así como planear en un calendario las fechas de desarrollo y salida al mercado.

RODAS Computación S.A. de C.V. es una empresa interesada en el desarrollo de línea de productos de software, por su parte desarrolla una LPS enfocada igualmente al dominio de domótica pero para hoteles (o inmótica) y accedió a compartir algunos de sus insumos de documentación para establecer correctamente las necesidades de los clientes de un producto para domótica. De acuerdo a la empresa y su encuesta realizada a 55 hoteles (anexo A), las principales características que tienen la posibilidad de ser incluidas en los primeros productos son las siguientes:

- **Luces automáticas:** La característica de iluminación se enfoca a la automatización de las habitaciones de las casas y hoteles, dicha cualidad busca satisfacer las expectativas de encender o apagar las luces de distintas maneras, para proporcionar un mayor confort a los habitantes y apoyar al economizar en el consumo de energía.

- **Detector de incendios:** La característica de detector de incendios se enfoca a ofrecer seguridad en las habitaciones de casas y hoteles, dicha cualidad busca ofrecer medios fiables para detectar a tiempo conatos de incendio y ser capaces de tomar las medidas necesarias para reducir los riegos generados por el fuego.

- **Solicitud de servicios:** La característica de solicitud de servicios se enfoca a aumentar el confort de los huéspedes de un hotel, facilitar la solicitud de algún servicio sin tener que ir hasta el lugar físico para hacer la solicitud y poder hacerlo desde la comodidad de la habitación.

- **Mensaje de bienvenida:** Esta cualidad de mensaje de bienvenida se orienta al ambiente hotelero y se enfoca en mejorar la satisfacción del cliente al momento de su llegada a la habitación, ya que se le dará una confortable bienvenida al momento de entrar a la habitación dando a conocer los servicios con los que cuenta el hotel.

- **Persianas automáticas:** Esta característica de persianas automáticas se enfoca en la automatización de las habitaciones de una casa o un hotel, dicha cualidad busca satisfacer las expectativas de poder abrir o cerrar de distintas maneras las persianas de una habitación, proporcionar un mayor confort a los habitantes y facilitar su control.

- **Energía inteligente:** Esta cualidad de energía inteligente se enfoca a apoyar específicamente el ahorro de energía dentro de las habitaciones de los hoteles debido a que en los mismos hay gastos excesivos de energía, esto beneficia la economía del hotel así como al desarrollo sustentable.

- **Acondicionamiento de temperatura:** Esta característica de acondicionamiento de temperatura se enfoca en satisfacer las expectativas de cómo encender, apagar o controlar la potencia de distintas maneras de aires acondicionados, calentadores, ventiladores o sistemas de calefacción para proporcionar un mayor confort a los habitantes.

- **Calentador de Agua Automático:** Esta cualidad de los calentadores de agua se enfoca a regular el uso de los calentadores de agua de una casa o habitación basándose en el

porcentaje de habitaciones ocupadas o la selección del usuario para proporcionar un mayor confort a los huéspedes y apoyar a economizar el consumo de gas.

El *plan de productos* describe las características de todas las posibles aplicaciones a generar por la línea de productos de software, categoriza las características que son parte de cada aplicación y las características variables que son sólo parte de algunas aplicaciones.

El plan de productos es el descrito a continuación:

Producto 1: Básico para el hogar

- Luces automáticas
- Detector de incendios

Producto 2: Avanzado para el hogar

- Luces automáticas
- Detector de incendios
- Persianas automáticas
- Acondicionamiento de temperatura

Producto 3: Básico para los hoteles

- Luces automáticas
- Detector de incendios
- Persianas automáticas
- Acondicionamiento de temperatura
- Energía inteligente
- Solicitud de servicios

Producto 4: Avanzado para los hoteles

- Luces automáticas
- Detector de incendios
- Persianas automáticas
- Acondicionamiento de temperatura
- Energía inteligente
- Solicitud de servicios
- Mensaje de bienvenida
- Calentador de agua automático

En términos académicos, la línea de productos de software desarrollada en esta tesis tiene un alcance domótico, es decir, para casas habitación, por lo que no se incluirán todas las características propuestas por la empresa ya que se enfocan a edificios muy grandes como hoteles. De cualquier forma la LPS desarrollada es capaz de incluir tales características en un futuro.

3.1.2. Ingeniería de requerimientos de dominio

El principal objetivo de esta etapa es la obtención y desarrollo de requerimientos de dominio comunes y variables y su documentación precisa. Es un proceso continuo en el cual se definen proactivamente los requerimientos para todos los productos que se desean generar en la LPS. Es importante identificar y documentar explícitamente la variabilidad interna y externa.

Definición de requisitos. Se espera que todos los productos generados tengan estas cualidades, por lo que se consideran requisitos comunes y serán parte de la plataforma o arquitectura de referencia: (1) las aplicaciones son capaces de controlar un conjunto establecido de dispositivos de hardware que a su vez harán lectura o modificaran valores del ambiente; (2) cada producto provee una interfaz de comunicación común y coherente para interactuar con sus servicios; (3) cada operación realizada se copia a la bitácora como medio de seguimiento del funcionamiento del producto; (4) el producto es independiente del dispositivo desde donde se realice la interacción

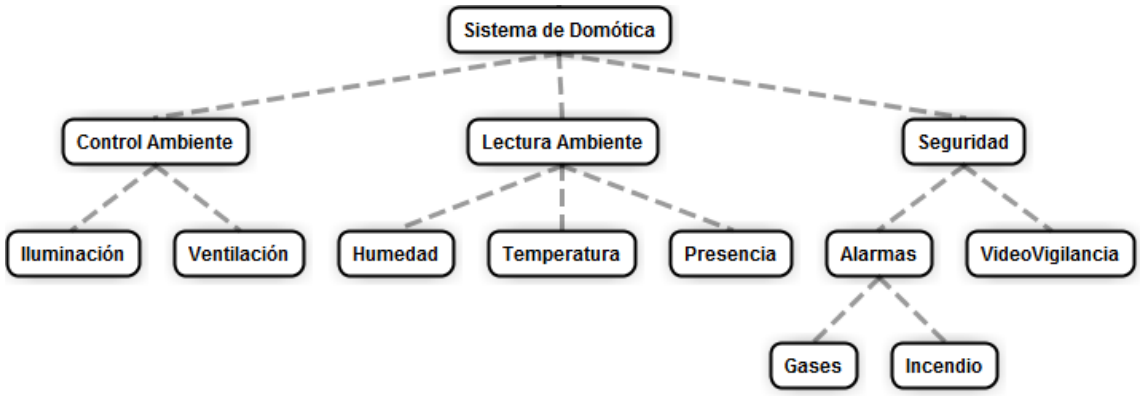

Figura 3.2: Diagrama CVL de variabilidad externa

entre el usuario y el sistema; y (5) el producto es independiente de la plataforma donde se ejecute, es decir, operar en cualquiera de los sistemas operativos más populares (Windows, Linux u OSX).

Para los requisitos variables se incluirán las características definidas en la etapa de *gestión del producto* con algunas otras que se explican más adelante.

Uno de los primeros pasos es modelar la variabilidad. Para modelar la variabilidad se utiliza la notación CVL, un estándar respaldado por la OMG (*Object Management Group*). Un diagrama CVL representa un árbol de características que incluirá la LPS. En las figuras 3.2 y 3.3 se representa la variabilidad de primer nivel desde dos diferentes puntos de vista.

La variabilidad externa de la figura 3.2 es la variabilidad percibida por el cliente, denota las características de las cuales un cliente elige un conjunto que satisfagan sus necesidades de control y presupuesto para crear un producto totalmente personalizado. En el diagrama se muestran al menos 8 ramas diferentes de características, si de las cuales se realizan combinaciones donde un producto tiene al menos dos características implementadas se tiene la posibilidad de elegir entre 247 combinaciones distintas para un producto. Sin embargo un producto con las características de sensor de temperatura y humedad no tendría mucho sentido, ya que no se tiene nada que controlar.

La variabilidad interna de la figura 3.3 es la variabilidad percibida por los desarrolladores, afecta a los artefactos de dominio y es invisible para los clientes. Por ejemplo, a un cliente no le interesa saber el gestor de base de datos que utilice un sistema, a él solo le interesa que el sistema

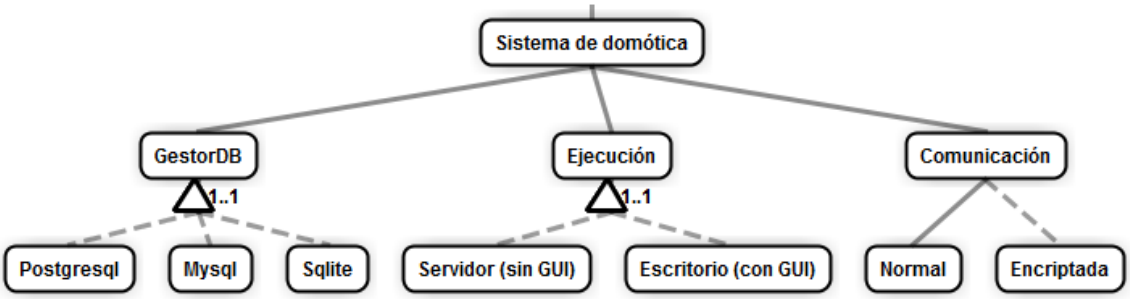

Figura 3.3: Diagrama CVL de variabilidad interna

funcione adecuadamente en los términos pactados de calidad. Según el presupuesto del cliente, es posible instalar el producto en la misma computadora que el cliente utiliza todos los días (la cual seguramente tiene un ambiente gráfico) o por sus hábitos no le agrada la idea de tener encendida su computadora todo el día y decide adquirir otro equipo que se encargue de ejecutar el sistema domótico las 24 horas del día sin necesidad de tener conectado un monitor. Estas decisiones afectan el desarrollo de la aplicación, sin embargo, el cliente no tiene que enterarse o no le interesan estas decisiones.

3.1.3. Diseño de dominio

El principal objetivo de esta etapa es diseñar y documentar la arquitectura de referencia, definir la estructura principal del software y su textura. En la arquitectura se ven reflejadas las decisiones del analista para incluir la variabilidad y los distintos requerimientos. La arquitectura de referencia generada provee las bases para realizar la implementación de la arquitectura y el diseño de la arquitectura de las aplicaciones o productos a generar. Se realizan las actividades tradicionales de diseño como la abstracción, modelado, simulación, prototipado y validación de los requerimientos.

Despliegue del sistema

El sistema domótico se conforma por dos aplicaciones de software, por una parte está el servidor de aplicaciones para domótica, el cual contiene todo lo necesario para controlar los

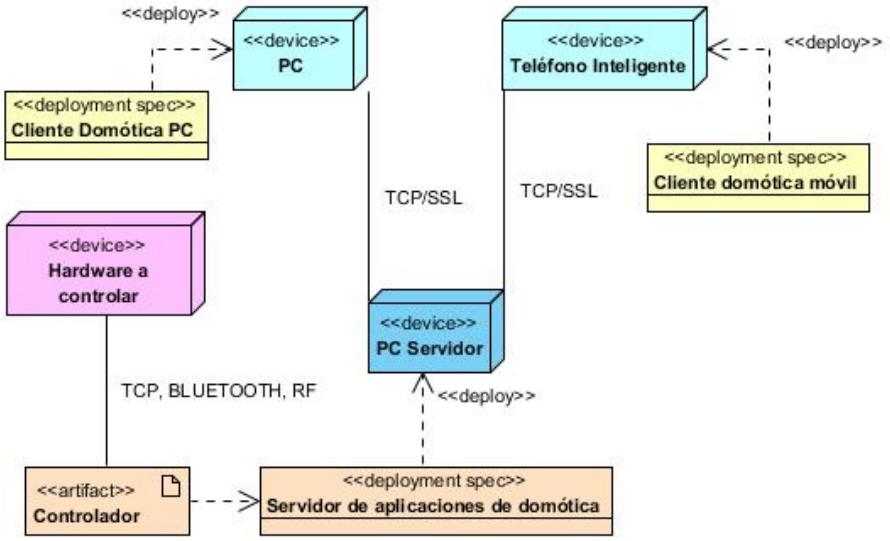

Figura 3.4: Diagrama de despliegue general del sistema domótico

dispositivos de hardware del ambiente domótico y es donde se ve reflejada la variabilidad; por otra parte está la aplicación cliente, esta es específica para un dispositivo, ya sea una computadora de escritorio o una laptop o algo más cómodo como un celular inteligente con sistema operativo Android o iOS. En la figura 3.4 se representa lo anteriormente dicho.

El sistema domótico contiene una serie de controladores, cada uno para un dispositivo de hardware específico, la comunicación entre un controlador y el hardware se da de distintas formas, ya sea a través de un cable Ethernet o WiFi (TCP/IP), o Bluetooth, señales de radio (RF), entre otras. También existe la posibilidad de acceder a hardware que controle a su vez otros dispositivos de hardware, por ejemplo, un controlador envía una petición a un dispositivo de hardware con capacidad de enviar señales de infrarrojo el cual es capaz de encender o apagar una televisión, cambiar de canal o subir el volumen, u otros dispositivos como un estéreo, un aire acondicionado o un teatro en casa.

En la figura 3.5 se presenta la misma estructura pero en capas, se pretende que el sistema domótico actúe como intermediario entre los dispositivos del hogar y las aplicaciones cliente que deseen controlarlos a través de un mismo protocolo y una interfaz común.

Figura 3.5: Capas del sistema domótico

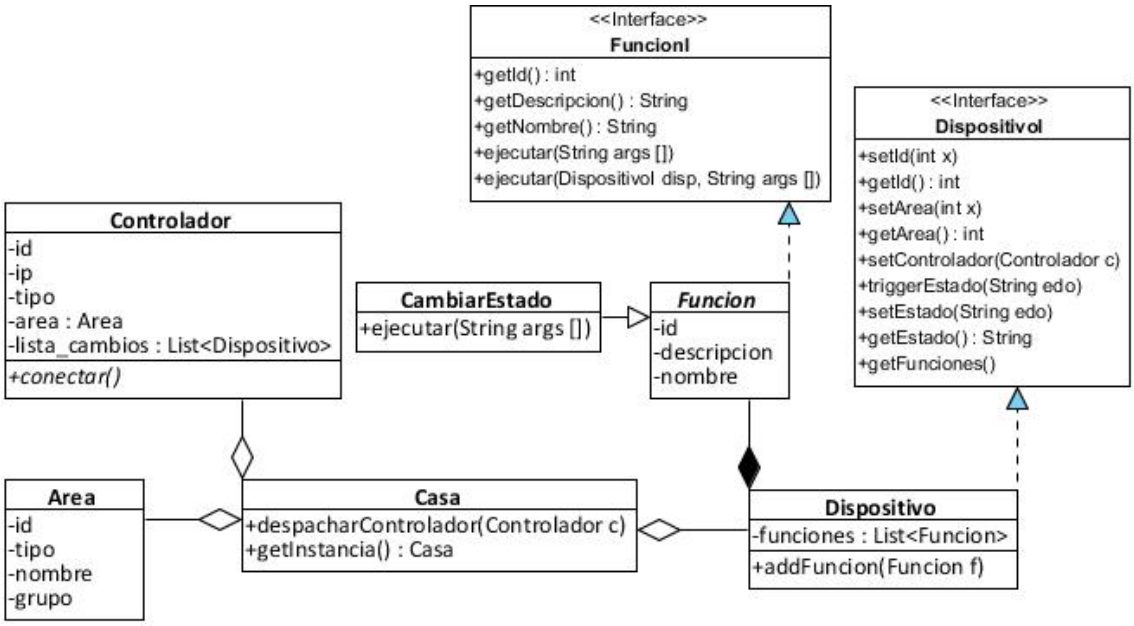

Figura 3.6: Diagrama de clases del paquete *modelos*

Diagramas de clases

El objetivo del sistema es encapsular el control de distintos tipos de dispositivos de hardware a través de una interfaz común y extensible, para garantizar esto a nivel de modelo, se planteó el diseño de la figura 3.6.

La clase *Casa* es la principal, hace uso del patrón *singleton* por lo que solo existe una instancia de ella, la clase tiene una serie de métodos que permiten conocer los dispositivos y

áreas que alberga. La clase *Area* es una abstracción de un espacio físico como un patio, un pasillo o una habitación, su objetivo es agrupar los dispositivos según el lugar donde se encuentren. La clase *Controlador* se encarga de interactuar directamente con el dispositivo de hardware del hogar, por esto *Controlador* tiene una lista de clases *Dispositivo*, su objetivo es que una clase *Luz* (por ejemplo) herede de ella, un mismo controlador es capaz de controlar distintos tipos de dispositivos, y por cada dispositivo específico se agrega una clase que lo represente, más adelante durante la ingeniería de aplicación se verá con más detalle. La clase *FuncionI* representa a un servicio disponible en un dispositivo (aquí se hace uso del patrón *Command*), todos los dispositivos tienen al menos una instancia de la clase *CambiarEstado* (excepto aquellos de solo lectura como humedad o presencia) porque es la operación básica a realizar sobre un dispositivo. Es posible agregar más tipos de comandos para tipos específicos de dispositivos, por ejemplo, una abstracción de una televisión tendría comandos como *CambiarCanal*, *SubirVolumen* o *ElegirFuente* entre otras funciones. La estructura final de la arquitectura de referencia (con las características implementadas) se hace transparente al usuario a través del protocolo de comunicación que se describe más adelante.

Los servicios que se mantienen escuchando peticiones se representan en la figura 3.7, existe un servidor para peticiones asíncronas sobre el protocolo TCP/IP, cada vez que un cliente se conecta (clase *Cliente*) este se mantiene conectado y mientras dura la conexión es capaz de realizar varias peticiones (una a una y después de autenticarse) y recibir actualizaciones de los estados de los dispositivos al momento en que ocurren (si el dispositivo lo permite). El servidor HTTP tiene el objetivo de recibir los mismos comandos y dar las mismas respuestas pero sobre la capa adicional de HTTP, orientado a peticiones síncronas, se realiza una conexión por cada petición y respuesta, si el cliente desea conocer el nuevo estado de un dispositivo entonces realiza una petición cada cierto tiempo (para esto mejor usar el primer servidor). Es útil para peticiones en las que se manipulan objetos en la base de datos o para inicio de sesión.

Protocolo de comunicación

Para la comunicación entre el servidor de aplicaciones domótico Intelidomo y las aplicaciones desplegadas en computadoras personales o dispositivos móviles, se diseñó un protocolo basado

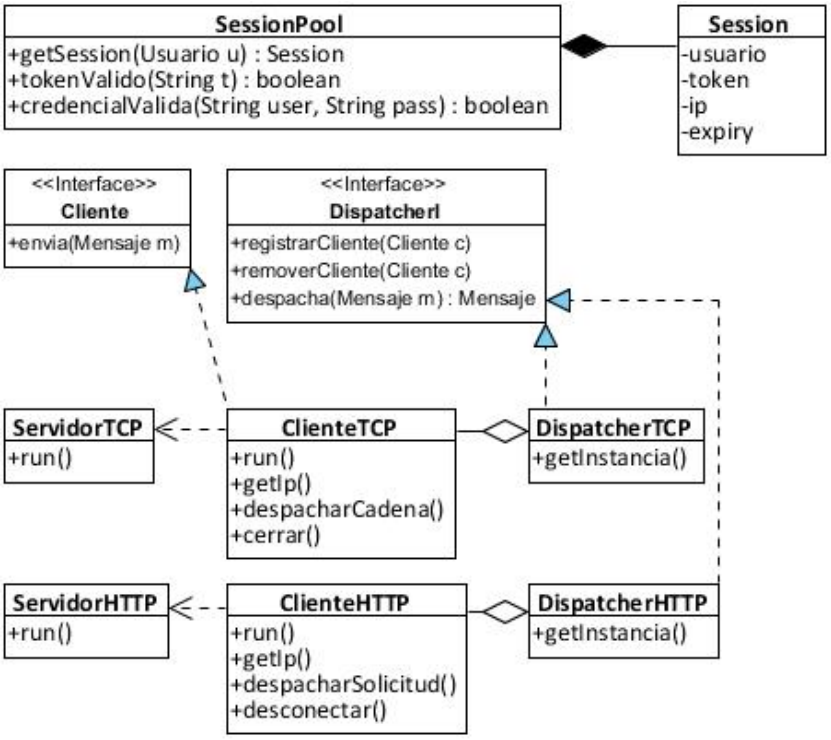

Figura 3.7: Diagrama de clases del paquete *server*

en XML que trabaja sobre canales de comunicación TCP/IP y HTTP. La primera versión del protocolo ofrece una interfaz de comunicación sencilla, autodescriptiva y común entre el servidor de aplicaciones de domótica Intelidomo y los dispositivos que se deseen conectar.

El protocolo se nombró *Intelidomo Communications Protocol* (IDCP). Sobre TCP/IP el intercambio de mensajes se hace a partir de cadenas XML separadas por un salto de línea vacío, la conexión TCP/IP permanece abierta mientras el cliente tenga abierta la aplicación cliente, esto para facilitar el aviso de eventos desde el servidor de aplicaciones a los clientes. Un cliente, después de autentificarse y solicitar la información de los dispositivos, obtiene un mensaje como el descrito en el código 3.1, a partir de entonces se conoce los tipos de dispositivos en el hogar. Para realizar el cambio de estado en un dispositivo se envía un mensaje como el del código 3.2 donde el tipo de mensaje es “1” (solicitud) y la operación a realizar es “UPDATE” (actualizar) indicando el nuevo estado en el atributo *estado* del dispositivo identificado con su tipo y número (*id*).

En la primera versión se encontraron algunas deficiencias y se realizó una nueva versión más

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<Mensaje tipo="2" accion="DOMOINFO">
  <Area id="1" tipo="HABITACION" descripcion="MI CUARTO"/>
  <Area id="2" tipo="HABITACION" descripcion="SALA"/>
  <Dispositivo categoria="ALARMA" subcategoria="GAS" id="1"
    tipoEstados="SINO_SL" estado="0" area="1"/>
  <Dispositivo categoria="SH" id="1" tipoEstados="DEC2"
    estado="7319" area="1"/>
  <Dispositivo categoria="ST" id="1" tipoEstados="DEC2"
    estado="2170" area="1"/>
  <Dispositivo categoria="SP" id="1" tipoEstados="SINO_SL"
    estado="0" area="1"/>
  <Dispositivo categoria="LUZ" id="1" tipoEstados="ONOFF"
    estado="0" area="1"/>
  <Dispositivo categoria="LUZ" id="2" tipoEstados="ONOFF"
    estado="0" area="1"/>
</Mensaje>
```

Listado 3.1: Respuesta en primera versión del protocolo

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<Mensaje tipo="1" accion="UPDATE">
  <Dispositivo categoria="LUZ" id="2" tipoEstados="ONOFF"
    estado="1" area="1"/>
</Mensaje>
```

Listado 3.2: Cambiar un estado en primera versión del protocolo

apegada al protocolo propuesto por [20]. La deficiencia principal es que existen dispositivos en los cuales no es suficiente (o muy adecuado) realizar cambios a través de un sencillo atributo *estado*, por ejemplo, en un ambiente donde se utilizan luces RGB y se desea controlar la iluminación con tonos exactos, es poco representativo hacerlo con un simple valor entero, se necesitan tres valores ya sea un entero (0-255) o una cadena representando un hexadecimal (00-FF). Lo mismo ocurre con una televisión, es mejor utilizar funciones para operaciones que se sabe tiene la televisión.

En el código 3.3 se muestra un ejemplo de la última versión del protocolo, se cambió el idioma de las etiquetas a inglés para darle forma de estándar, se agregaron etiquetas que describen mejor el estado y tipo de estado aceptados, así como funciones que el cliente es capaz de ejecutar y permiten agregar más de un argumento. En el código 3.4 está el ejemplo de una llamada hecha para encender luces (basándose en la respuesta del código 3.3). No es necesario indicar el nombre

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<Message type="2" action="DOMOINFO">
  <Area id="1" type="ROOM" descripcion="MY ROOM"/>
  <Area id="2" type="ROOM" descripcion="LIVING ROOM"/>
  <Device category="ALARM" subcategory="GAS" id="1" area="2">
   <State type="YES_NO" readOnly="YES" value="0" />
  </Device>
  <Device category="ALARM" subcategory="FIRE" id="2" area="2">
   <State type="YES_NO" readOnly="YES" value="0" />
  </Device>
  <Device category="HUM" id="1" area="1">
    <State type="PERCENT" readOnly="YES" value="73.2"/>
  </Device>
  <Device category="TEMP" id="1" area="1">
   <State type="CELCIUS" readOnly="YES" value="21.5"/>
  </Device>
  <Device category="LIGHT" id="1" area="1">
   <State type="ON_OFF" readOnly="NO" value="1"/>
   <Function name="setEstado" instanceId="1">
    <Arg name="state">
   </Function>
   <Function name="encenderLuz" instanceId="2"/>
   <Function name="encenderLuz" instanceId="3"/>
  </Device>
  <Device category="LIGHT" id="2" area="1">
   <State type="ON_OFF" readOnly="NO" value="0"/>
   <Function name="setEstado" instanceId="4">
    <Arg name="state">
   </Function>
   <Function name="encenderLuz" instanceId="5"/>
   <Function name="encenderLuz" instanceId="6"/>
  </Device>
</Message>
```

Listado 3.3: Información de dispositivos en segunda versión del protocolo

de la función, solo su identificador de instancia. Se están encendiendo las dos luces encontradas, en el primer caso a través del método *cambiarEstado* indicando en el argumento el nuevo valor; en el segundo caso se hace directamente con la función *encenderLuz* que no necesita ningún argumento.

En el caso del servidor HTTP, este es capaz de atender peticiones GET o POST y la respuesta de igual manera siempre será un mensaje XML del protocolo IDCP, previamente descrito. El servidor HTTP siempre cierra la conexión automáticamente al responder, por lo que sólo es

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<Message type="1">
 <Execute>
  <Function instanceId="1">
   <Arg name="state" value="1">
  </Function>
  <Function instanceId="5"/>
 </Execute>
</Message>
```

Listado 3.4: Encendiendo luces en segunda versión del protocolo

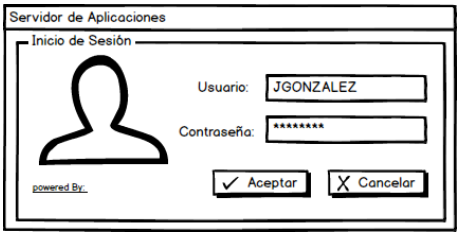

Figura 3.8: Bosquejo pantalla *inicio de sesión*

posible recibir un mensaje por conexión. Las peticiones HTTP son mejores para ciertas funciones, como iniciar sesión o cargar los dispositivos.

Interfaz de usuario

El usuario será capaz de controlar al servidor de aplicaciones Intelidomo a través del celular o computadora, estas aplicaciones de control son desarrolladas por separado, fuera de la línea de productos de software ya que no tienen variabilidad, es por eso que es necesario el protocolo IDCP que ofrece una forma de comunicación común, no variable y extensible. Algunas de las pantallas propuestas están en las figuras 3.8, 3.9 y 3.10.

3.1.4. Implementación del dominio

Como se ha mencionado antes, el lenguaje de desarrollo seleccionado ha sido DeltaJ. El código fuente de la plataforma se organizó en módulos delta (es necesario para que el compilador reconozca las clases; clases o recursos que son estáticos -como imágenes- se agregan directamente

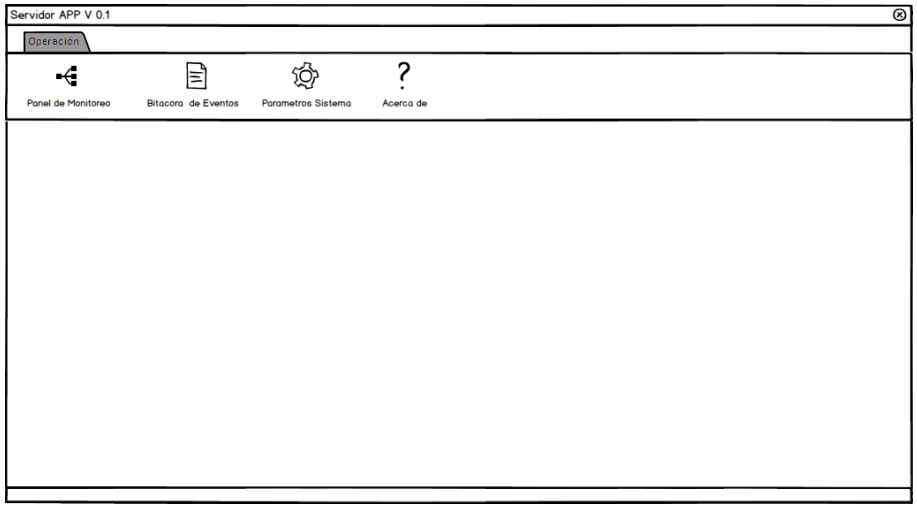

Figura 3.9: Bosquejo pantalla de bienvenida

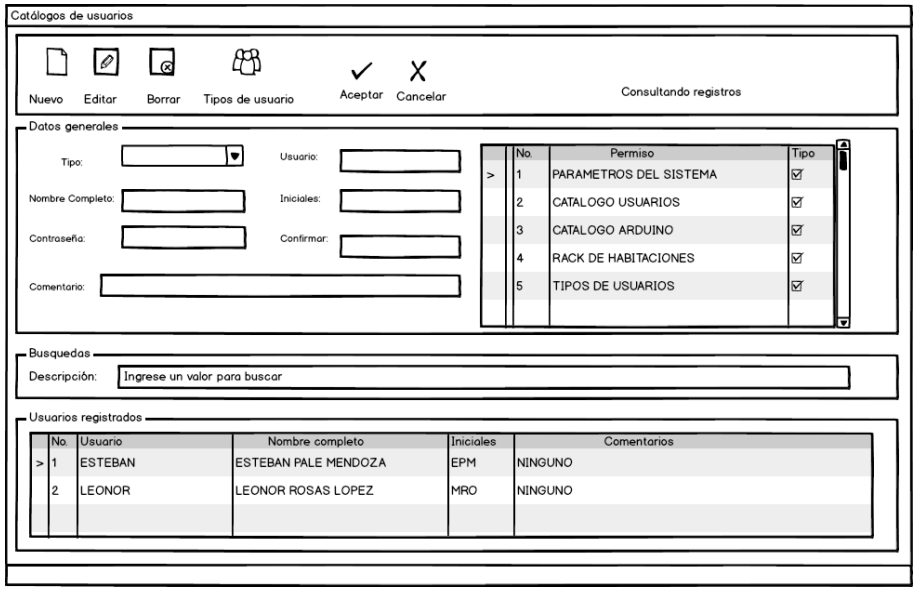

Figura 3.10: Bosquejo pantalla *administración de usuarios*

en un paquete Java puro). Estos módulos que generan la plataforma se aplican en todos los productos, es necesario hacer mención explícita de la característica *base*, los módulos son: *dBase*, *dDataBase*, *dModelos* y *dServer*.

El módulo *dBase* agrega lo necesario para iniciar la aplicación: la clase principal, clases de bitácora, clases de carga de configuración, entre otras; el módulo *dDataBase* agrega clases base necesarias para crear una conexión de base de datos utilizando las interfaces que provee Java utilizando como intermediario el protocolo de *JDBC*; el módulo *dModelos* agrega las clases o entidades modelo principales como *Casa* o *Area*; por último el módulo *dServer* agrega las clases necesarias para iniciar un servidor TCP/IP y manejar múltiples conexiones concurrentes.

Tal como se ve en el código 3.5 el módulo delta *dBase* tiene únicamente instrucciones de agregar código fuente (nada de modificar o eliminar), ya que se crea la plataforma desde cero y esta será la misma para todas las aplicaciones.

3.2. Ingeniería de aplicación

La ingeniería de aplicación es el proceso de la ingeniería de Líneas de Productos de Software en el cual las aplicaciones de la línea de producción se desarrollan y se generan reutilizando artefactos de la ingeniería de dominio y explotando la variabilidad de la linea de productos. Algunas de las fases de este proceso se automatizaron al desarrollar un configurador de Líneas de Productos de Software del cuál se habla en el siguiente capítulo.

3.2.1. Ingeniería de requerimientos de aplicación

El objetivo de esta fase es obtener y documentar los artefactos de requerimientos para una aplicación particular y al mismo tiempo reutilizar, tanto como sea posible, los artefactos de requerimientos del dominio.

Durante esta fase se detallaron los requerimientos obtenidos en el dominio y se organizaron y consolidaron en una única tabla según el orden de implementación de los productos. El configurador de Líneas de Productos de Software implementa este subproceso y de ello se habla

```
delta dBase {
  adds{
    package aplicacion;
    import db.Gestor;
    import modelos.Casa;
    public class Inicio{
      public Inicio(){...}
      public void constructor(){...}
      public static void main(String args[]){...}
    }
  }
  adds{
    package aplicacion;
    public class Config{
      public static final long RUN_ID = System.currentTimeMillis();
      public static int SERVIDORCOMANDOS_PORT = 4490;
      public static String LOG_DIR = "C:\\var\\";
      public static int LOG_LEVEL = 2;
      public static void cargarValores(){...}
    }
  }
  adds{
    package aplicacion;
    import java.util.logging.*;
    import java.io.IOException;
    public class Bitacora{
      public final static int MESSAGE= 1, NOTICE = 2, WARNING = 3,
          ERROR = 4, FATAL_ERROR = 5;
      private static int min_level = 2;
      static Logger logger = Logger.getLogger("DomoLog");
      static{...}
      public static void setMinLevel(int level){...}
      private static void addToLogger(int level, String txt, int
       ↪  idp){...}
      public static void add(int level, String txt, int idp){...}
    }
  }
}
```

Listado 3.5: Módulo delta dBase

más adelante.

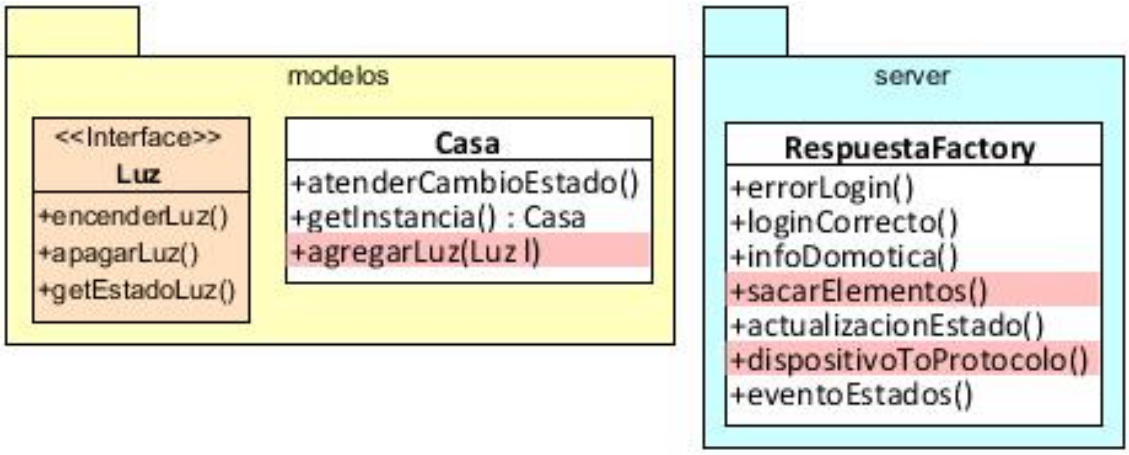

Figura 3.11: Clases de la característica *Luces*

3.2.2. Diseño de la aplicación

El objetivo de esta etapa es producir la arquitectura específica para cada aplicación. La arquitectura de la aplicación es una especialización de la arquitectura de referencia desarrollada en el diseño de dominio.

Para dar vida a cada característica se plantearon diversos cambios en la arquitectura de referencia. Por ejemplo, para la característica de iluminación, se agregó una clase abstracta (figura 3.11) para abstraer la funcionalidad de las luces. De igual manera es necesario realizar algunos cambios en algunos métodos, se marcan con el mismo color los métodos de las clases que será necesario modificar para soportar la funcionalidad.

El configurador de Líneas de Productos de Software implementa este subproceso y de ello se habla más adelante

3.2.3. Implementación de la aplicación

El objetivo de esta fase es desarrollar aplicaciones capaces de probarse y llevarse al mercado después de asegurarse que cumplen con las expectativas de calidad. La implementación de la aplicación provee un diseño e implementación detallada de los componentes específicos del producto y los configura con las variantes adecuadas desarrolladas durante la implementación de la plataforma para dar vida a nuevas aplicaciones o productos.

Para dar vida a las diferentes características del dominio de domótica e implementarlas, se generaron los siguientes módulos delta cuyo mismo nombre describe la característica que permite implementar.

- dLuces
- dPresencia
- dTemperatura
- dHumedad
- dAlarma

En el código 3.6 se muestra la fuente del módulo delta *dLuces*. Lo que hace en primer lugar es agregar una interfaz llamada *Luz*, de la cual heredarán las clases controladores de las implementaciones de *hardware* que hagan las aplicaciones. Después se procede a modificar la clase *Casa* para que esta reconozca a los dispositivos *Luz*, se agrega un mapa de luces donde el identificador de cada luz es un número consecutivo conforme se van añadiendo luces, se agrega un método *registrarLuz()*, se modifica el método *atenderCambioEstado()* que es el encargado de saber a qué objeto de qué tipo enviarle una petición de cambiar su estado, básicamente se le enseña a la clase *Casa* a manejar los objetos *Luz*. De igual manera cuando se modifica la clase *server.RespuestaFactory* al modificar los métodos *sacarElementos()* y *Dispositivo2Protocolo* lo que se hace es permitir a las clases encargadas de comunicarse con las aplicaciones cliente a comunicar el nombre y estado de cada luz, así como de entender cuando se hace alguna petición sobre alguna luz.

Los demás módulos delta funcionan de manera semejante, agregando y modificando lo necesario para dar vida a la característica que describen.

En estos momentos ya se es capaz de incluir estas características en una aplicación, sin embargo, para que sean útiles es necesario agregar clases de control de implementaciones de *hardware* específicas.

```
delta dLuces {
  adds{
    package modelos;
    public interface Luz extends DispositivoI{
      public String getEstadosLuz();
      public void setEstadoLuz(int edo);
      public int getEstadoLuz();
    }
  }
  modifies modelos.Casa{
    adds public Map<Integer,Luz> mapaLuces = new HashMap<Integer,Luz>();
    adds private static int luz_count = 1;
    adds public void registrarLuz(Luz l){...}
    modifies atenderCambioEstado(String tipo, int id_disp, int
     ↪ nuevoEdo){
      if(tipo.equals(Constantes.LUZ)){...}
        return original(tipo,id_disp,nuevoEdo);
      }
  }
  modifies modelos.Area{
      adds public List<Luz> listaLuces = new ArrayList<Luz>();
      adds public void agregarLuz(Luz l){
        listaLuces.add(l);
      }
  }
  modifies server.RespuestaFactory{
    adds import modelos.Luz;
    modifies sacarElementos(Mensaje m, Area area){
      for(Luz luz : area.listaLuces){...}
      original(m,area);
    }
    modifies DispositivoToProtocolo(Dispositivo dx){
      if(dx instanceof Luz){...}
      return original(dx);
    }
  }
}
```

Listado 3.6: Módulo delta dLuces

En el código 3.7 se agrega una clase capaz de controlar la implementación de hardware desarrollada basada en la Raspberry Pi nombrada *DomoPi*. Esta clase tiene lo necesario para hacer la conexión TCP/IP con el dispositivo *DomoPi* y enviar/recibir de lectura o control. Ya específicamente, en el código 3.8 se agrega una clase que abstrae una funcionalidad del dispositivo de hardware, en el caso de las luces agrega una clase *LuzImp* que implementa *Luz* y está vinculada

con la clase controladora del dispositivo *DomoPi*, el delta agrega a la clase *Domopi* lo necesario para entender los mensajes correspondientes al estado de las luces y también para enviar los comandos que se encargan de encender o apagar una luz, identificada por un número, según la cantidad de luces conectadas al dispositivo de hardware.

```
delta dDomopi {
  adds{
    package modelos.domopi;
    import intelidomo.idml.Constantes;
    import modelos.DispositivoControlador;
    import modelos.Casa;
    import java.io.*;
    import java.net.*;
    public class Domopi extends DispositivoControlador{
      private Conexion conn;
      private long tiempo_reconn = 10000;
      private int intentos_reconn = 0;
      public Domopi(){
        setTipo("Domopi Controller");
      }
      public void constructor(){...}
      @Override
      public void conectar(){...}
      private void entenderCadena(String str){...}
      private void entenderPedazo(String key,String val){}
      public void enviarComando(String cmd){...}
      public class Conexion implements Runnable{
        String comando;
        Socket socket;
        PrintWriter out;
        BufferedReader in;
        @Override
        public synchronized void run(){...}
        private void esperarConexion(){...}
      }
    }
  }
  modifies db.Gestor{
    adds import modelos.domopi.Domopi;
    modifies despacharDispositivo(String tipo,ResultSet rs){
      if(tipo.equals("Domopi")){
        Domopi dodisp = new Domopi();
        dodisp.setId(rs.getInt("id_disp"));
        dodisp.setArea(rs.getInt("id_area"));
        dodisp.setIP(rs.getString("ip"));
        dodisp.constructor();
        return dodisp;
      }
      return original(tipo,rs);
    }
  }
}
```

Listado 3.7: Módulo delta dDomopi

```
delta dDomopiLuces {
  adds{
    package modelos.domopi;
    import intelidomo.idml.Constantes;
    import modelos.Dispositivo;
    import modelos.Luz;
    import java.util.logging.*;
    public class LuzImp extends Dispositivo implements Luz{
      private final Domopi padre;
      private final char cid;
      public LuzImp(Domopi drpi, char id){...}
      @Override
      public String getEstadosLuz(){
        return Constantes.ON_OFF;
      }
      @Override
      public int getEstadoLuz(){
        return estado;
      }
      @Override
      public void setEstadoLuz(int edo){
        if(edo==Constantes.ON) padre.enviarComando("SET LUZ"+cid+" ON");
        else if(edo==Constantes.OFF) padre.enviarComando("SET LUZ"+cid+"
          ↪  OFF");
      }
      public char getIdChar(){ return cid; }
    }
  }
  modifies modelos.domopi.Domopi{
    adds Map<Character,LuzImp> mapa_luces = new
      ↪  HashMap<Character,LuzImp>();
    modifies entenderPedazo(String key,String val){
      if(key.startsWith("LUZ")) {
        ...
      }
      original(key,val);
    }
  }
}
```

Listado 3.8: Módulo delta dDomopiLuces

Capítulo 4

Resultados

La finalidad de este capítulo es presentar los resultados obtenidos fruto de la aplicación del marco de trabajo para LPS [2], los árboles de características CVL y del paradigma de programación orientada a deltas en el campo de domótica. También se muestran los resultados de la experimentación con microcontroladores programables, módulos electrónicos (de sensores y actuadores) y plataformas de desarrollo que permiten crear dispositivos propios capaces de solucionar algunas necesidades del ambiente domótico.

Para facilitar la identificación del sistema domótico y sus aplicaciones, se nombró como Intelidomo al conjunto de productos y aplicaciones desarrolladas del dominio de domótica. Por lo cual, el objetivo de Intelidomo es ofrecer soluciones de confort y seguridad el usuario, permitiéndole visualizar y controlar aspectos de su hogar desde el dispositivo con el que más se sienta cómodo.

4.1. Productos de domótica generados

Una vez implementada la arquitectura de referencia, se implementaron una a una las características del ambiente domótico, de uno a uno los dispositivos con que se contó (aquellos que se desarrollaron y los de otros proveedores). Esto se tomó en cuenta para definir y nombrar los productos a generar por la línea de productos de software.

En necesario recordar que el sistema Intelidomo está divido en dos partes principales, por un

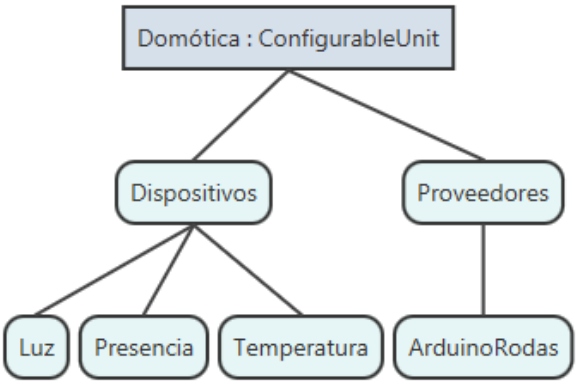

Figura 4.1: Árbol de características CVL del producto *Rodas*

lado se tiene al servidor domótico, donde está presente la arquitectura de referencia y el poder de combinación de la variabilidad, y por otro lado, las aplicaciones cliente no variables que utilizan la interfaz del servidor domótico. Tales aplicaciones sólo conocen el tipo de dispositivo a controlar o tipo de sensor y su estado, nunca se enteran de qué controlador o intermediarios se utilizaron para lograr obtener el valor.

Las diferencias entre productos se perciben en el servidor domótico, y en las interfaces o funcionalidades que éste ofrece. También es posible que exista más de una aplicación cliente que explote diferentes funcionalidades del servidor u ofrezca diferentes formas de manipularlo, pero se consideran parte del producto ofrecido y no como nuevos productos, ya que por sí solas no son capaces de algo funcional.

Los diagramas de clases y diagramas de características presentados en esta sección, se generaron por el configurador desarrollado del que se habla más adelante, en la siguiente sección.

El primer producto generado se titula *Rodas*, esto porque el primer dispositivo que se comenzó a implementar en la LPS fue el Arduino programado por la empresa Rodas Computación S.A. de C.V. (el cual se encontraba en su primera versión). El árbol de características CVL de la figura 4.1 corresponde a tal producto. El dispositivo es capaz de controlar una luz, realizar lectura de presencia, así como de temperatura. Al momento de implementar este dispositivo, sus funciones se encontraban limitadas ya que aún estaba en etapa de desarrollo. La empresa también desarrolló un simulador capaz de interpretar los mismos comandos y responder de la misma manera en

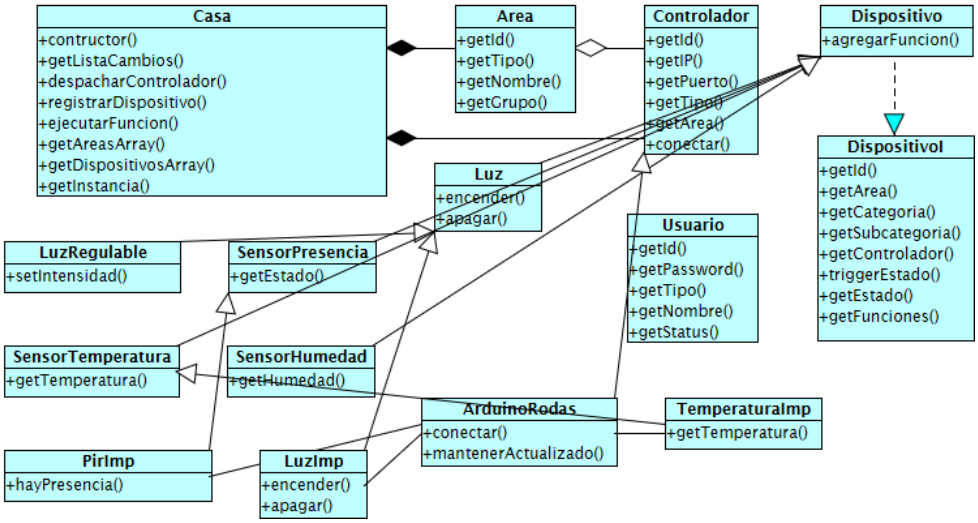

Figura 4.2: Arquitectura del producto *Rodas*

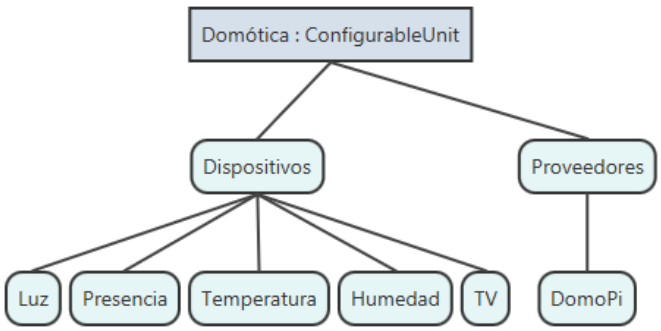

Figura 4.3: Árbol de características CVL del producto *Domopi*

que lo hace el dispositivo físico. Se trabajó sobre este simulador ya que no fue posible para la empresa prestar su prototipo físico.

La arquitectura de referencia se vio modificada y una parte de la arquitectura final del producto se visualiza en la figura 4.2. Se requirió agregar una clase *ArduinoRodas* heredada de *Controlador*, la cual se encarga de gestionar la conexión con el dispositivo físico, enviar los comandos de acuerdo a su protocolo y entender la respuesta recibida. Las clases *LuzImp*, *PirImp* y *TemperaturaImp* son implementaciones específicas para este controlador de los dispositivos de luz, sensor de presencia y sensor de temperatura. El controlador crea una instancia de cada implementación y registra las referencias en la instancia de *Casa*.

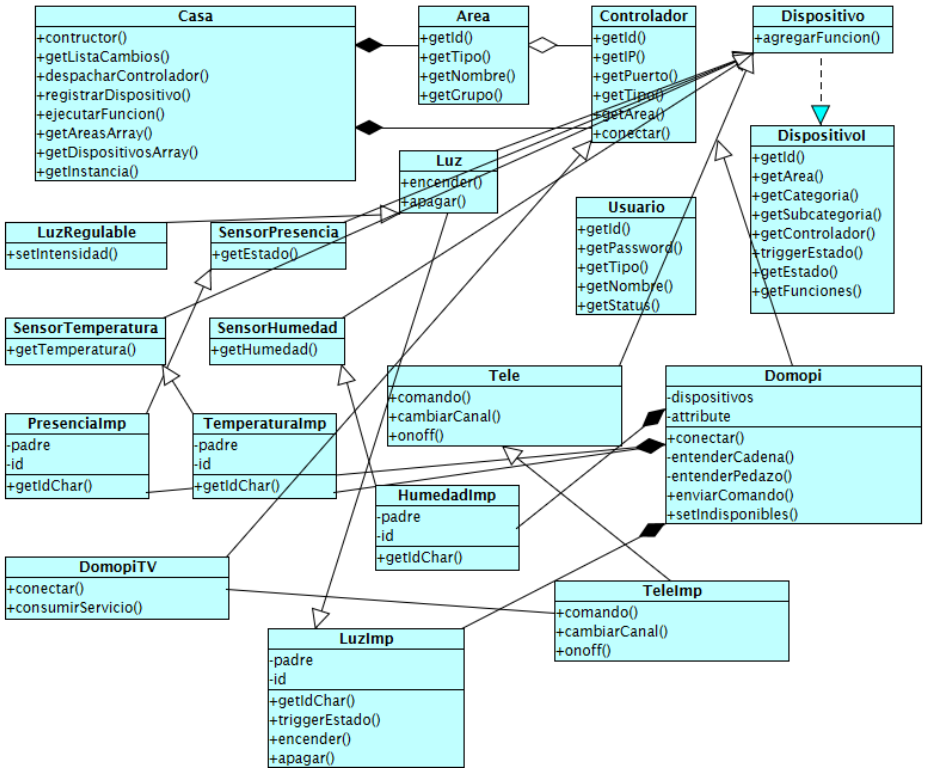

Figura 4.4: Arquitectura del producto *Domopi*

El segundo producto generado se llama *Domopi*, esto por el segundo dispositivo desarrollado e implementado en la LPS. El árbol de características CVL de la figura 4.3 corresponde a este producto. El dispositivo es capaz de controlar diversas luces, hacer lectura del estado de sensores de presencia, sensores de temperatura y sensores de humedad; también de enviar señales de infrarrojo para el control de televisiones.

La arquitectura del producto quedó tal como se muestra en la figura 4.4. Se agregó la clase *Domopi* (heredando de *Controlador*), la cual se encarga de gestionar la conexión con el dispositivo físico. Las clases *LuzImp*, *PresenciaImp*, *TemperaturaImp* y *HumedadImp*, actúan como las implementaciones de los dispositivos de luz, sensor de presencia, sensor de temperatura y sensor de humedad. El controlador *Domopi*, al preguntar el estado de los sensores y actuadores al dispositivo físico y obtener respuesta, crea instancias de las clases antes mencionadas y registra las referencias en el sistema. En el caso del controlador *DomopiTV*, este se encarga de realizar las peticiones HTTP al servidor del dispositivo Domopi, el cual provee una interfaz sobre el

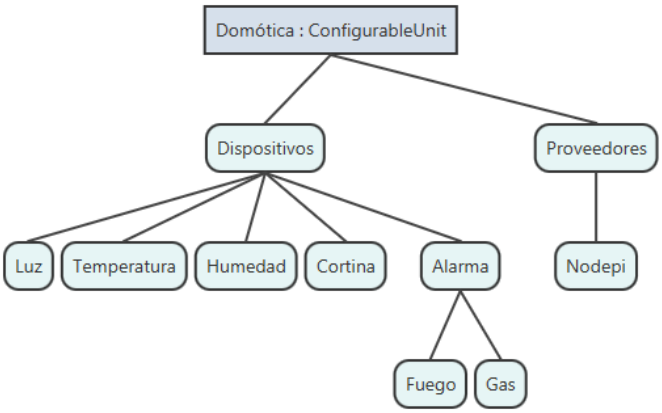

Figura 4.5: Árbol de características CVL del producto *Nodepi*

protocolo HTTP que permite indicar que comando enviar a la televisión a través de infrarrojo.

El tercer producto generado es llamado *Nodepi*, de acuerdo al tercer dispositivo desarrollado e implementado en la LPS. El árbol de características CVL para este producto es el que se muestra en la figura 4.5, implementa las características de control de luces, lectura de temperatura y humedad, control de cortinas, y por último, activación de alarmas de fuego y gases peligrosos. Estas características se implementan por tres tipos diferentes de dispositivos físicos.

La arquitectura del producto se presenta en la figura 4.6. La clase *NodepiSensors* es el controlador dedicado a conectarse a un dispositivo con diferentes sensores (temperatura, humedad, flama y gases), crea instancias de las clases que implementan a los sensores y los registra en el sistema. El controlador pregunta al dispositivo el estado de los sensores una vez por segundo. La clase *CortinaController* es el controlador dedicado al dispositivo de control de cortinas, crea una instancia de *CortinaImp*, la cual conoce los comandos a enviar para establecer una posición específica en la cortina física. La clase *NodepiDimmableLights* es el controlador para las luces de intensidad regulable, crea dos instancias de *LuzDimp*, la cual es una implementación de *LuzRegulable*. *LuzDimp* indica al controlador los comandos a enviar para establecer una intensidad específica en la luz física.

El cuarto producto integra las funcionalidades de los tres productos anteriores, esto se refleja en el diagrama de la figura 4.7 y en la arquitectura del producto de la figura 4.8. Este producto se considera completo por incluir la funcionalidad de los 3 dispositivos utilizados en productos

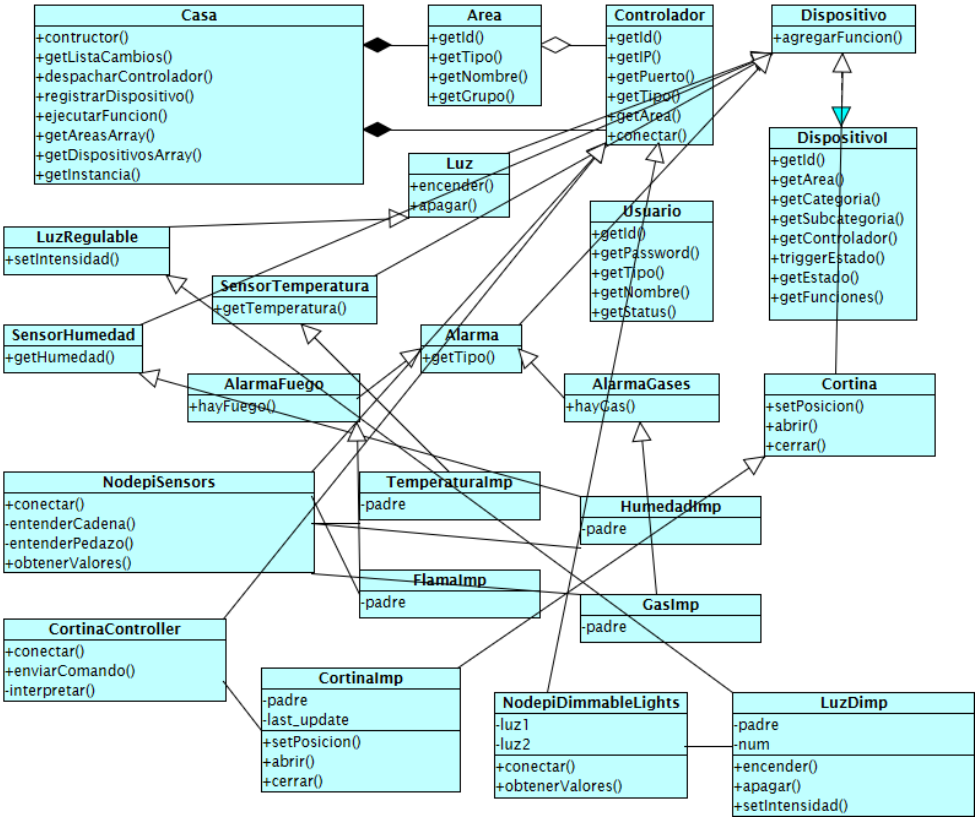

Figura 4.6: Arquitectura del producto *Nodepi*

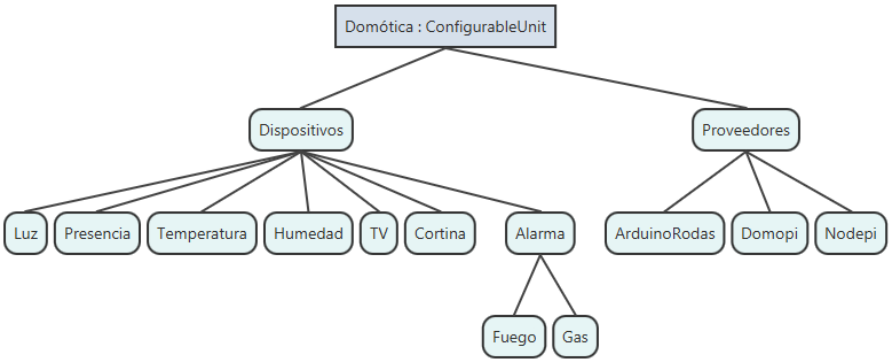

Figura 4.7: Árbol de características CVL del producto *Completo*

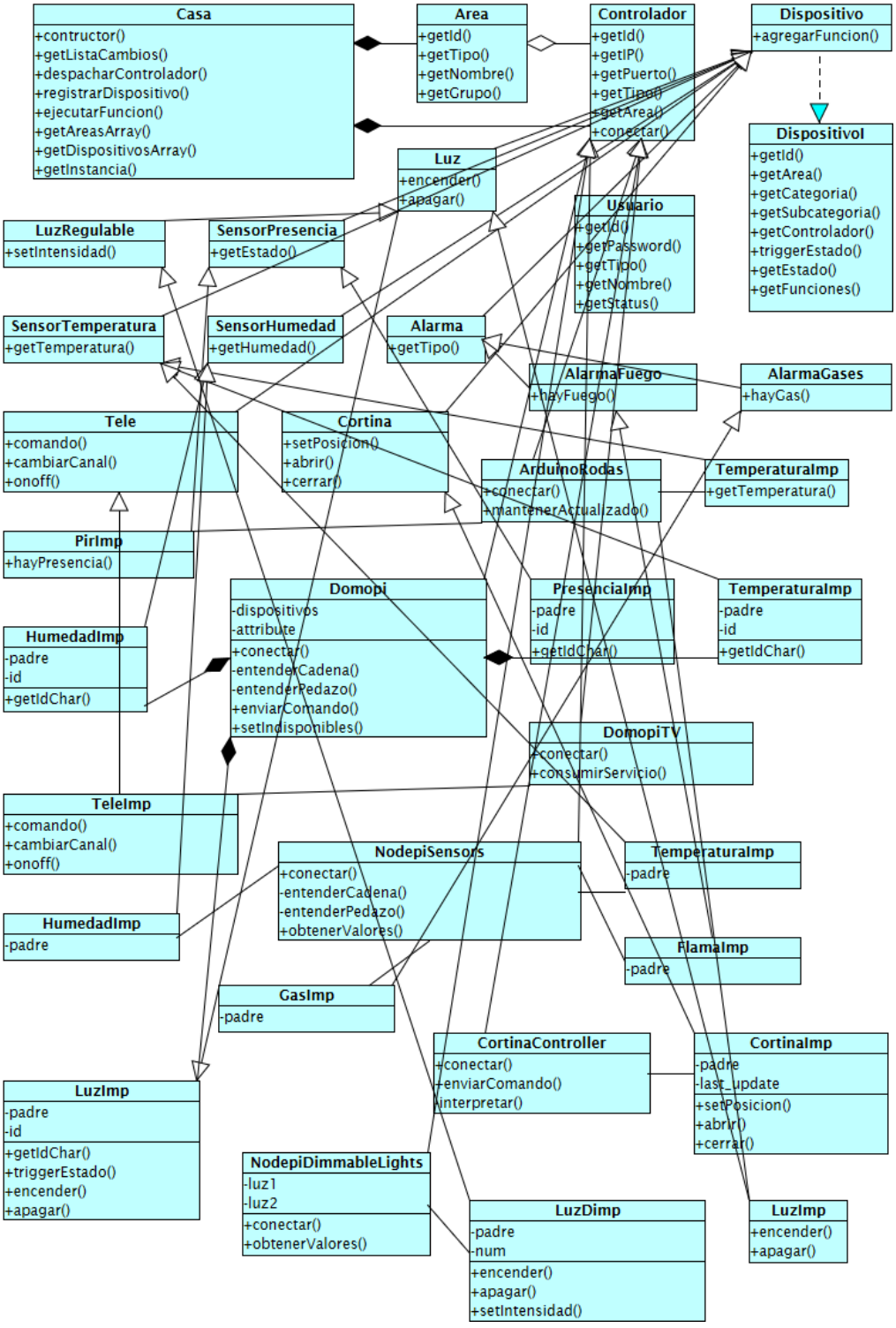

Figura 4.8: Arquitectura del producto *Completo*

anteriores, sin embargo aún se consideran otros dos productos, parecidos al completo, pero con características extras.

Tabla 4.1: Características y productos de la LPS

Característica	**Rodas**	**Domopi**	**Nodepi**	**Completo**	**Plus**	**Custom**
Luces	✓	✓	✓	✓	✓	✓
Presencia	✓	✓	✗	✓	✓	✓
Temperatura	✓	✓	✓	✓	✓	✓
Humedad	✗	✓	✓	✓	✓	✓
TV	✗	✓	✗	✓	✓	✓
Cortinas	✗	✗	✓	✓	✓	✓
Alarma fuego	✗	✗	✓	✓	✓	✓
Alarma gases	✗	✗	✓	✓	✓	✓
Alarma gases	✗	✗	✓	✓	✓	✓
Soporte Arduino Rodas	✓	✗	✗	✓	✓	✗
Soporte Domopi (Raspberry)	✗	✓	✗	✓	✓	✓
Soporte Nodepi (NodeMCU)	✗	✗	✓	✓	✓	✓
Soporte de luces MiLight	✗	✗	✗	✗	✓	✓
Control por voz	✗	✗	✗	✗	✓	✓

En la tabla 4.1 se muestra el resumen de los productos de la LPS y las características que implementan. El producto *plus* se caracteriza por incluir permitir el control de luces MiLight (dispositivos de control de focos RGB manufacturado por la empresa MiLight) y por agregar la funcionalidad de aceptar comandos en lenguaje natural (para aprovechar las API voz a texto que algunas plataformas ofrecen). El producto *custom* (personalizado) se diferencia del producto *plus*

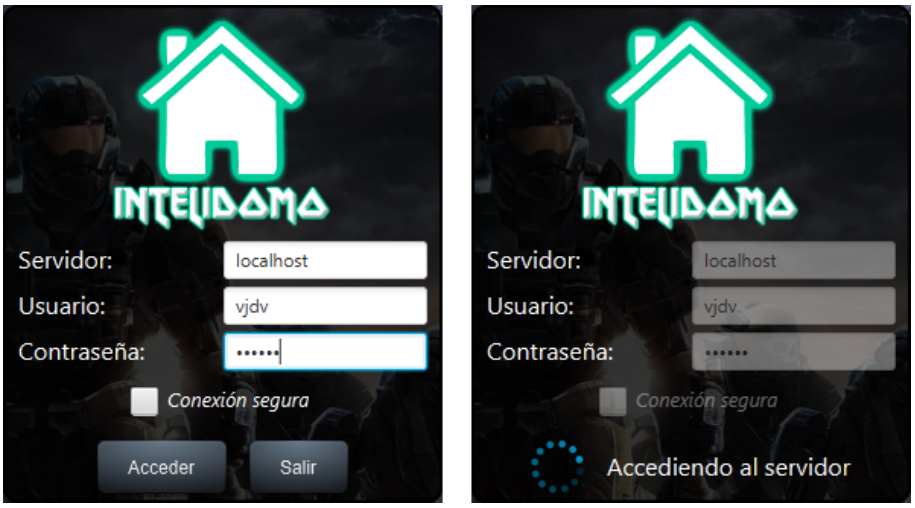

Figura 4.9: Pantalla de inicio de sesión de aplicación de escritorio

por excluir al controlador *ArduinoRodas*, ya que no se considera útil para fines prácticos por no tener el dispositivo físico.

4.1.1. Aplicación de escritorio

Se optó por desarrollar la aplicación de escritorio en la plataforma JavaFX, esto ya que cuenta con un mejor motor gráfico, da la posibilidad de cambiar el aspecto de los componentes a través de reglas de estilo CSS (mucho más flexible que *Swing*), ofrece nuevos contenedores de diseño y controles avanzados como reproductor de vídeo y renderizador de objetos 3D.

Al abrir la aplicación se muestra la pantalla de inicio de sesión de la figura 4.9, en esta parte el usuario deberá introducir la IP o dirección del servidor Intelidomo al que se desea conectar, seguido por su clave de usuario y contraseña. El usuario tiene la posibilidad de elegir si desea establecer una conexión segura o no (cifrada sobre SSL o normal sobre TCP). El puerto por defecto al que la aplicación intentará conectarse al acceder al servidor, será 4480 para peticiones HTTP y 4490 para conexiones TCP, en cambio si la conexión es segura, el servidor se conectará al puerto 4481 para peticiones HTTP y 4491 para conexiones TCP.

Cuando el usuario da clic al botón acceder, la aplicación envía los datos del usuario a través de una petición POST HTTP, todas las recibidas están en formato XML de acuerdo al protocolo

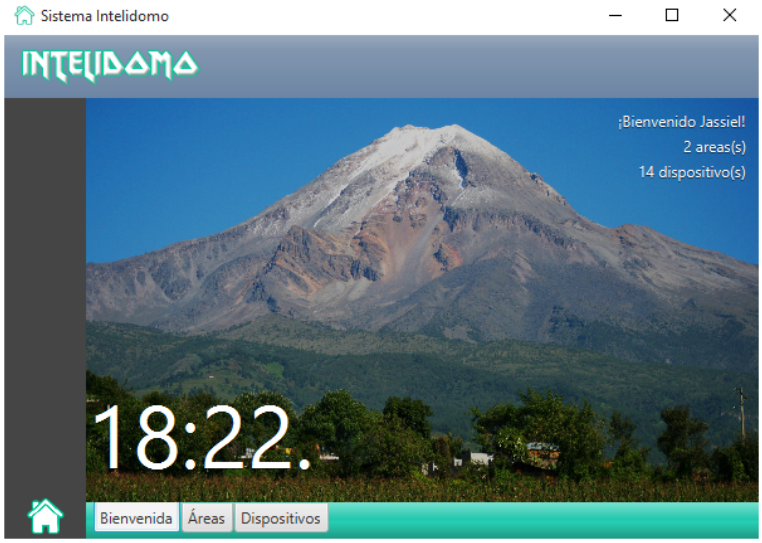

Figura 4.10: Pantalla de bienvenida de la aplicación de escritorio

de comunicación establecido. En la interfaz se le comunicará al usuario si ocurre algún error de conexión o sus datos de acceso son incorrectos. Si sus datos son correctos, se le concede el acceso y la aplicación recibe un *token*, el cual utiliza para realizar nuevas peticiones, esto mejora la seguridad ya que evita enviar el usuario y contraseña en todos las conexiones que se hagan, y el *token* es sólo válido para la IP del cliente conectado. Por medio de otra petición HTTP se cargan toda la información de los dispositivos disponibles en el hogar, una vez hecho esto, se carga la interfaz principal de la aplicación.

La aplicación comienza mostrando la pantalla de bienvenida de la figura 4.10. El cuerpo de la interfaz se compone de una cabecera donde se encuentra el logo de la aplicación, una barra lateral a la izquierda con un ícono de una casa, y un contenedor con pestañas para controlar la aplicación.

La aplicación se conecta al servicio TCP del servidor y mantiene la conexión abierta mientras se encuentre abierta la aplicación, esto le permite al servidor enviar actualizaciones de estado en tiempo real a la aplicación, por lo que siempre se mostrarán estados actualizados sin necesidad de que el usuario solicite actualizarlos o la aplicación tenga que pedirlos. Mientras la conexión sea válida, el ícono de la casa permanece en color verde, pero si se llega a perder la conexión, éste cambiará a rojo. La aplicación intentará reconectarse al instante, pero el tiempo que tarde en

Figura 4.11: Pestaña de áreas de la aplicación de escritorio

intentar incrementa con el número de intentos fallidos, para evitar carga innecesaria en la red.

La pestaña de bienvenida sólo muestra un resumen de la información domótica de la casa, la hora y una imagen de fondo a gusto del usuario.

La siguiente pestaña corresponde a las áreas, tal como se ve en la figura 4.11. Muestra un ícono por cada área, además de indicar su nombre o etiqueta asignada. Al dar clic sobre una, se despliega una barra lateral a la derecha indicando los dispostivos que pertenecen a dicha área.

La siguiente pestaña corresponde a la de dispositivos, como se muestra en la figura 4.12. En esta, se muestran todos los dispositivos, sin un orden específico, sin importar su área.

Por cada dispositivo se muestra un pequeño panel que contiene el ícono del dispositivo, el nombre de dispositivo, número de dispositivo y una etiqueta mostrando su estado o un control que permite cambiar el estado del dispositivo.

El control de los dispositivos es fácil e intuitivo, el control se adapta según el tipo de estado del dispositivo. Es posible encender o apagar una luz con un único botón, si la luz permite establecer su intensidad entonces se muestra un botón deslizable, o en cambio, si la luz permite establecer un color específico, entonces se muestra un control de elección de color. Todo se adapta al dispositivo y a su tipo de estado.

Existen dispositivos más complejos para los que un sólo botón no basta para controlarlos,

Figura 4.12: Pestaña de dispositivos de la aplicación de escritorio

Figura 4.13: Controles avanzados en la aplicación de escritorio

por ejemplo, una televisión. Cuando esto ocurre, se muestra un botón que permite abrir un panel lateral derecho con los controles completos para tal dispositivo. En la figura 4.13 se muestran el panel personalizado para una televisión que simula un control remoto convencional.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<Message type="1" action="EXECUTE">
  <Execute>
    <Function instanceId="0">
      <Arg name="tecla" value="KEY_POWER"/>
    </Function>
  </Execute>
</Message>
```

Listado 4.1: Ejecutando una función en el servidor

De igual manera, la manera de hacer las peticiones al servidor también cambia para estos dispositivos. De acuerdo al protocolo, en lugar de solicitar un cambio de estado, se solicita la ejecución de una funcion. Por ejemplo, para encender o apagar una televisión, es necesario enviar el mensaje XML del listado 4.1.

4.1.2. Aplicación móvil

Para aprovechar la característica que ofrecen los productos *plus* y *custom* de aceptar comandos en lenguaje natural, se desarrolló un cliente en forma de aplicación nativa para teléfonos Android. Esta aplicación se considera una característica complaciente, no reemplaza a la aplicación de escritorio, es una aplicación complementaria y ofrece una manera distinta de acceder a las funciones del sistema Intelidomo.

En las figuras 4.14 y 4.15 se muestran las capturas de pantalla que ilustran las operaciones básicas que la aplicación es capaz de realizar. Al abrir la aplicación se muestra una interfaz sencilla (figura 4.14a) que consta de una cabecera indicando el título de la aplicación, un menú, dos botones y una contenedor vacío semitransparente. Al tocar el botón, el dispositivo comienza el proceso de reconocimiento de voz y abre el diálogo de la figura 4.14b, durante este periodo de tiempo el usuario dicta un comando, y después de procesar el sonido, el dispositivo móvil

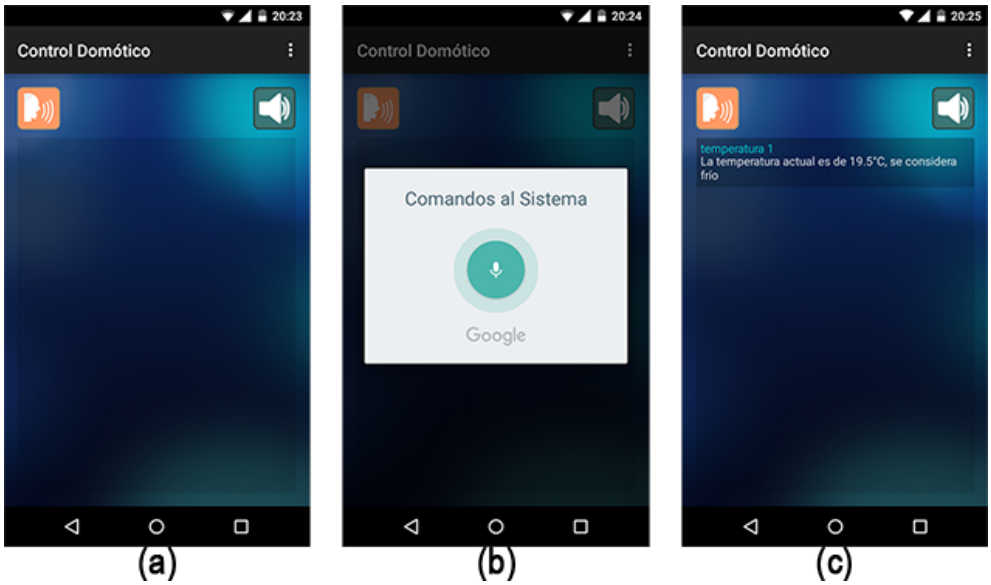

Figura 4.14: Pantallas de la aplicación móvil

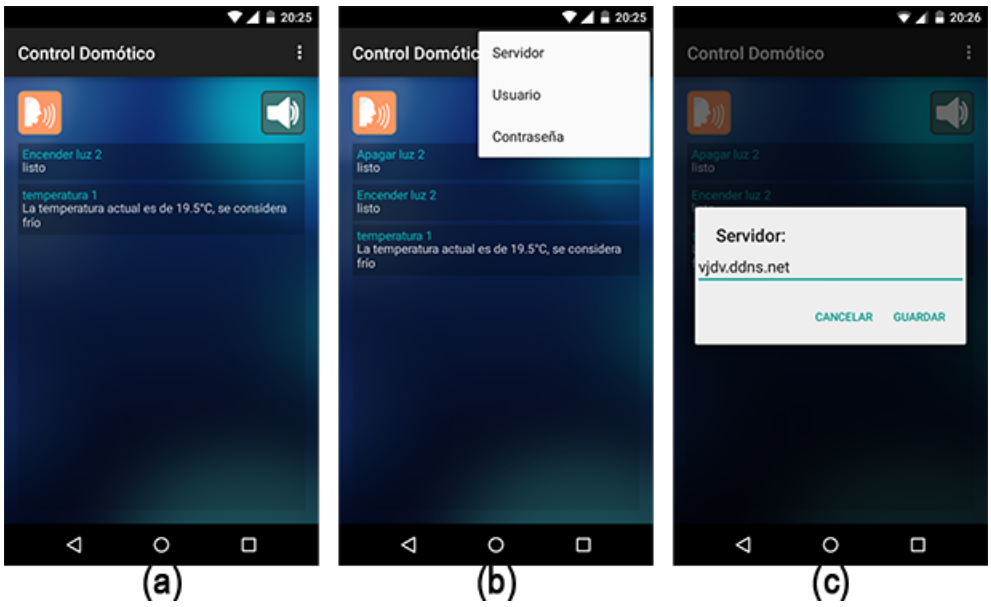

Figura 4.15: Pantallas de la aplicación móvil (parte 2)

envía el texto resultante al servidor Intelidomo a través de una petición POST HTTP. El servidor interpreta el texto, realiza la acción que se le pidió (si se requiere) y devuelve una respuesta en formato XML (sobre HTTP). En la figura 4.14c, se aprecia a la aplicación mostrando la leyenda *la temperatura actual es de 19.5°C, se considera frío* como resultado de dictar el comando *temperatura 1*.

Por el momento, los comandos aceptados son:

- **[temperatura]+número de sensor:** Devuelve en lenguaje natural el valor del sensor de temperatura con el número indicado, además de una estimación de cómo podría percibirse.
- **[temperatura de]+nombre de área:** Devuelve en lenguaje natural el promedio de temperatura indicada por los sensores que se encuentren en el área solicitada.
- **[temperatura promedio]:** Devuelve en lenguaje natural el promedio de temperatura de todos los sensores disponibles en el hogar.
- **[humedad]+número de sensor:** Devuelve en lenguaje natural el valor de la humedad relativa del sensor con el número solicitado, además de indicar si el valor se encuentra dentro de los valores recomendados.
- **[humedad promedio]:** Devuelve en lenguaje natural el valor promedio de todos los sensores de humedad relativa disponibles en el hogar.
- **[encender|apagar][luz]+número de luz:** Enciende o apaga la luz a la que le corresponda el número solicitado. Devuelve el texto "*¡Listo!*" al realizar la acción.
- **[encender|apagar][luces de]+nombre de área:** Enciende o apaga las luces que se encuentren en el área solicitada.
- **[encender|apagar][todas las luces]:** Enciende o apaga todas las luces que se encuentren disponibles en el hogar.
- **[encender|apagar][televisión]+número de televisión:** Envía el comando de encender o apagar a la televisión a la que le corresponda el número solicitado. Devuelve el texto "*¡Listo!*" al realizar la acción.

- **[televisión]+número de televisión+[canal]+número de canal:** Envía a la televisión a la que le corresponda el número solicitado los comandos necesarios para conseguir sintonizar el canal pedido.

La aplicación utiliza el API de TTS de Android (*Text To Speech* para dictar al usuario la respuesta obtenida, esto para ofrecer una respuesta en el mismo formato en el que se da el comando y evitar un sentimiento de *palabras al aire*. Si esto no es del agrado del usuario, al tocar el segundo botón el gráfico de éste cambiará al de una parlante enmudecida, de esta manera las respuestas ya no serán dictadas al usuario.

Los comandos dictados y respuestas obtenidas durante la sesión se van apilando en el contenedor semitransparente (figura 4.15a). En la figura 4.15b se muestra el menú de la aplicación expandido, permite modificar la configuración de la aplicación, ya sea la dirección del servidor al que se enviarán las peticiones, o las credenciales del usuario (clave y contraseña). En la figura 4.15c se visualiza un cuadro de diálogo sobre la aplicación que permite cambiar la dirección del servidor Intelidomo después de elegir esa opción en el menú.

4.2. Configurador reutilizable de LPS

DeltaJ ofreció una gran flexibilidad al configurar, definir y generar una serie de productos partiendo de un conjunto de características y módulos delta que definen las reglas de combinación de los trozos de código fuente desarrollados; esto permitió realizar la implementación de la línea de productos de software para domótica de una manera rápida e incremental. Sin embargo, de acuerdo al marco de trabajo utilizado (figura 3.1), es necesario cubrir actividades referentes a la ingeniería de aplicación, ya que aunque se tiene definida la documentación de la ingeniería de dominio (requerimientos, casos de uso, diagramas de clases, plan de pruebas), hace falta todavía trabajar sobre la documentación referente a la ingeniería de aplicación, es decir, contar con una documentación específica para cada uno de los productos generados, reutilizando los insumos desarrollados anteriormente en la ingeniería de dominio.

Se consideraron dos posibles soluciones. La primera solución considerada fue crear por

cada producto un nuevo documento y en él copiar y pegar fragmentos de la documentación de ingeniería de dominio que concuerda con el producto, sin embargo esto se consideró tedioso por requerir esfuerzo extra innecesario al dar mantenimiento o actualizar la documentación (por ejemplo, si se actualiza un requerimiento perteneciente a la arquitectura de referencia, sea cual sea la razón -errores tipográficos, cambios en las necesidades-, se tiene que buscar ese mismo fragmento en todos los documentos y reemplazar manualmente el texto, haciendo el proceso monótono y complicado de actualizar), y en contra del paradigma de Líneas de Productos de Software al no haber algún grado de automatización (y por lo tanto, ningún beneficio o ventaja en comparación con el desarrollo tradicional).

Se optó por la segunda opción: desarrollar un configurador secundario que abarque las áreas no cubiertas por DeltaJ y que flexibilizara la generación de documentación de cada aplicación o producto, completando el proceso sugerido por el marco de trabajo. Este configurador es una solución para cualquier proyecto de línea de productos de software que utilice el mismo marco de trabajo utilizado (o alguna adaptación de él) y esté orientado al desarrollo composicional como el efectuado por DeltaJ. La interfaz se hizo de una manera muy intuitiva, el usuario poco debe conocer sobre los estándares utilizados o el formato del archivo al guardar o recuperar proyectos.

4.2.1. Interfaz del configurador

Al abrir el configurador se muestra la pantalla de la figura 4.16, al dar clic en el menú *archivo*, se muestran las opciones disponibles para comenzar a utilizar el configurador, las cuales son: (1) crear un nuevo proyecto desde cero; (2) abrir un proyecto realizado anteriormente; (3) guardar el proyecto sobre el que se esté trabajando; (4) guardar el proyecto sobre el que se esté trabajando en una ubicación diferente a la actual; (5) importar un archivo CVL externo, el configurador es totalmente compatible con archivos creados por la herramienta *CVL 2 Tool* propuesta originalmente en [21]; (6) importar un archivo XMI, diferentes herramientas de modelado UML permiten exportar sus proyectos en formato XMI, el configurador es totalmente compatible con el estándar XMI 2.1 y de momento sólo es capaz de interpretar diagramas de clases; y (7) salir de la aplicación.

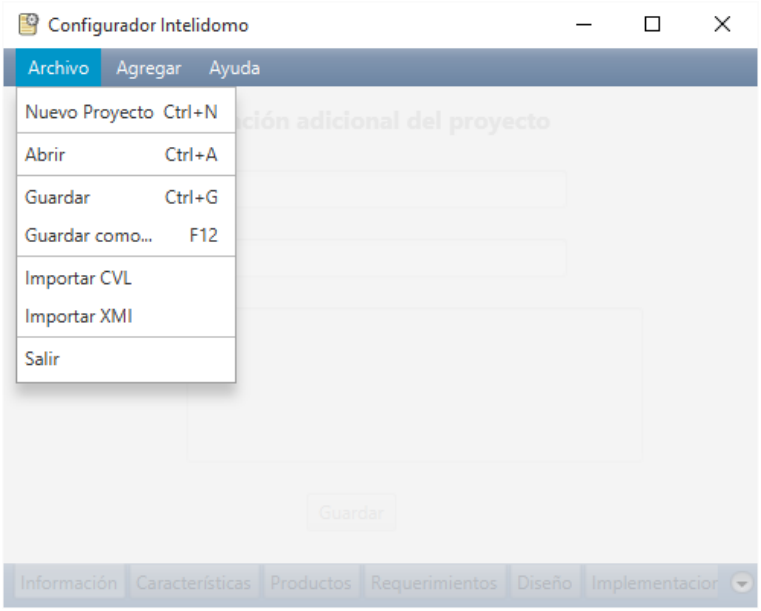

Figura 4.16: Pantalla principal del configurador

Cuando se selecciona la opción de crear un proyecto nuevo o abrir uno existente, se habilitan y preparan las pestañas de la interfaz (si es necesario se llenan con la información recuperada del proyecto). Las pestañas tienen las siguientes etiquetas: información, características, productos, requerimientos, diseño, implementación, pruebas y exportación. Su nombre y orden es de acuerdo al marco de trabajo.

La pestaña de información, visible en la figura 4.17, está dedicada a capturar información básica de la LPS, tal como es su nombre corto o apodo (utilizado como prefijo al nombrar los archivos generados por el configurador), su nombre largo (nombre completo del proyecto), así como una introducción o resumen de lo que se pretende con el proyecto. Esta información se exporta en todos los documentos generados por el configurador, sea ingeniería de dominio o ingeniería de aplicación.

La siguiente pestaña es la de *características* (figura 4.18), su objetivo es facilitar al usuario definir las características disponibles para los productos en la LPS. Esta pestaña es muy importante, ya que la información capturada aquí se utilizará en todas las demás etapas. Para facilitar la implementación de esta etapa, se desarrolló por separado una aplicación, a la que se nombró *CVL Maker*, imitando las propiedades de la herramienta *CVL 2 Tool* de [21], siendo capaz de crear

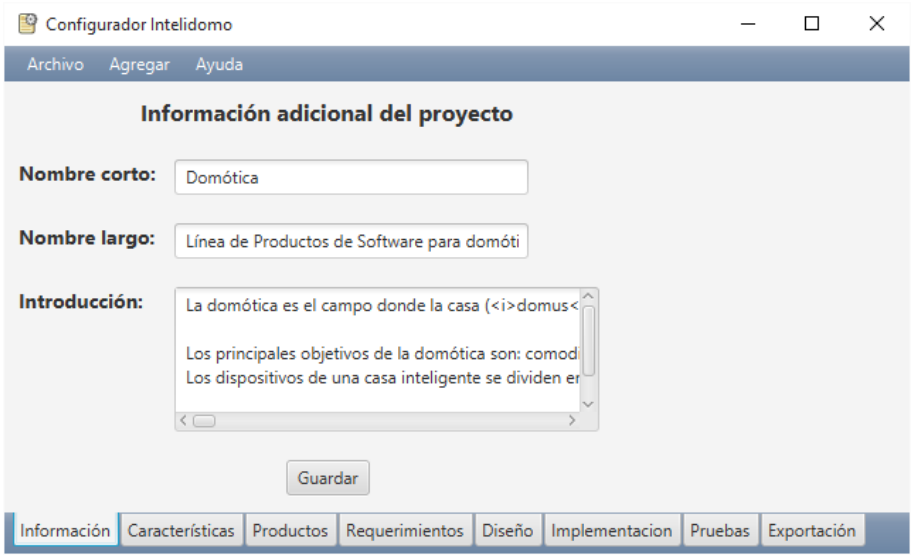

Figura 4.17: Pestaña de información de la LPS

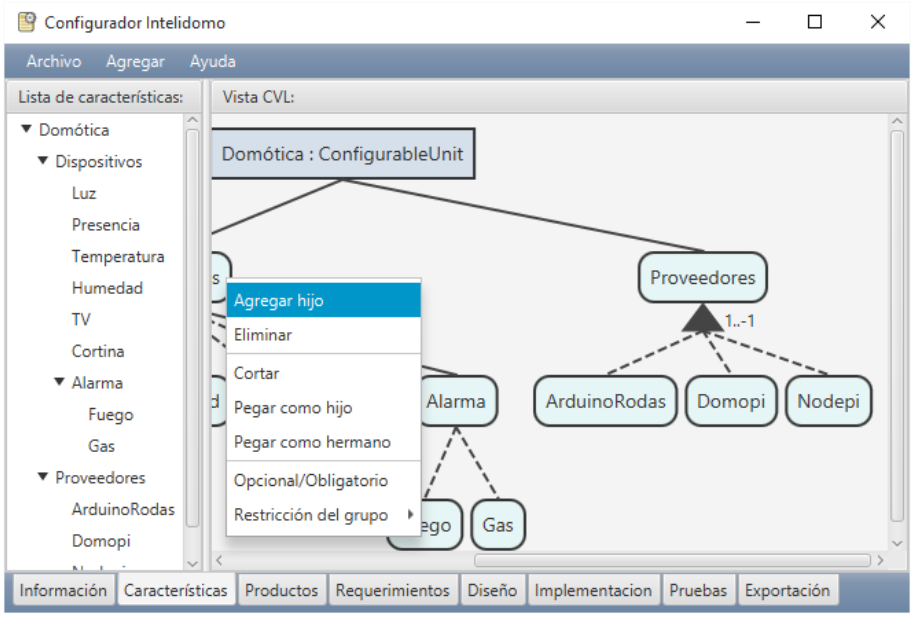

Figura 4.18: Pestaña de características de la LPS

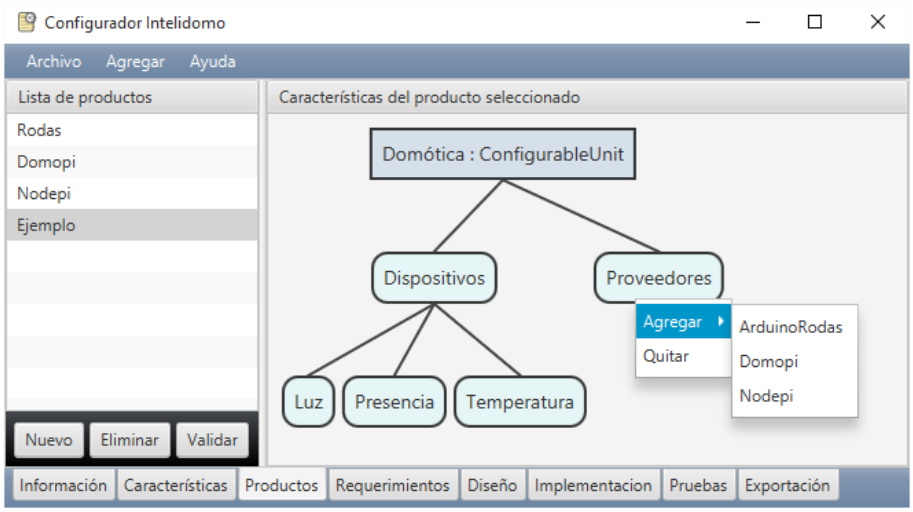

Figura 4.19: Pestaña de productos de la LPS

diagramas CVL en cuestión de minutos y ser mutuamente compatibles. Luego los componentes de *CVL Maker* se reutilizaron en el configurador, dándole la habilidad de editar y crear diagramas basados en el estándar CVL. La edición se hace a través de menús contextuales que permiten agregar nodos al árbol, eliminar nodos, establecer una característica como opcional u obligatoria, definir una restricción de grupo, así como cortar/pegar nodos unos en otros.

Para facilitar aún más la lectura de las características, al ir agregando o quitando elementos en el CVL, la lista de la izquierda se va actualizando mostrando de manera vertical y agrupada todos los elementos. También se van guardando en un *ArrayList* en el modelo del proyecto y se ve reflejado al guardar (véase la etiqueta *Características* del XML del listado 4.3).

La pestaña de *productos* (figura 4.19) permite definir de manera rápida los productos que se van a ofrecer y las características que cada uno de ellos tendrá. Al dar clic sobre el botón con la etiqueta *Nuevo*, se abre un cuadro de diálogo preguntando el nombre para el nuevo producto. Al introducir el nombre, este se agrega a la lista de la izquierda, al dar clic sobre él, el usuario es capaz de seleccionar y agregar las características específicas para el producto. De igual manera se muestra un editor CVL, con la diferencia de que esta vez no se permiten las mismas operaciones de la pestaña anterior, sólo se permiten las opciones de agregar/quitar hijos y siendo únicamente los que se establecieron anteriormente en la pestaña de características.

No es posible cambiar una característica a opcional, ya que aquí es la definición de las

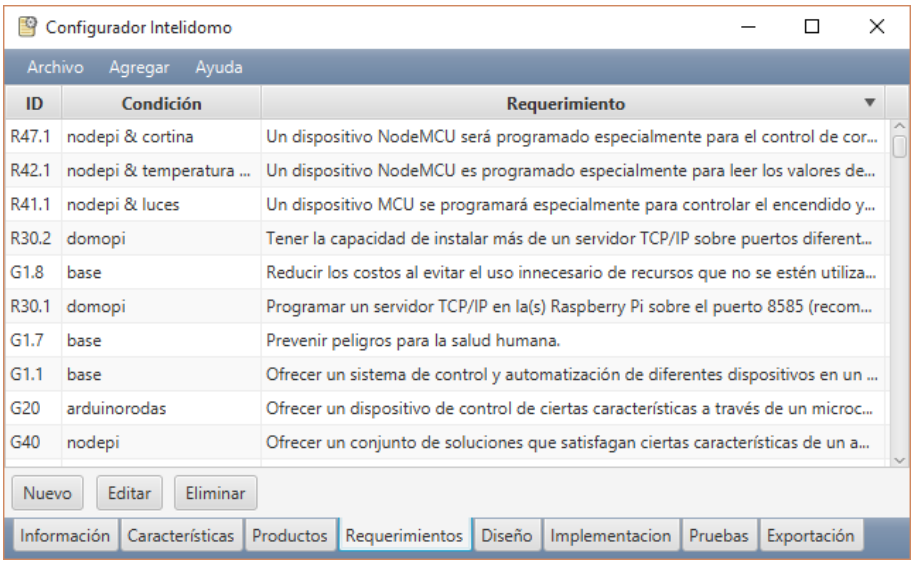

Figura 4.20: Pestaña de requerimientos de la LPS

características que obligadamente sí tendrá un producto. También están disponibles los botones de *validar* un producto (que las características agregadas no incumplan una restricción de grupo o de obligatoriedad definida anteriormente) y de *eliminar* un producto (quitarlo de la lista y del proyecto).

En el XML del proyecto del listado 4.3, los productos se ven reflejados en la etiqueta *Productos*, donde por cada producto se crea una etiqueta *Producto* y dentro se encuentran definidas las características elegidas para ese producto.

La siguiente pestaña (figura 4.20) permite definir todos los requerimientos deseados para la línea de productos de software. El alcance de la aplicación es al momento de requerimientos textuales, cada requerimiento capturado tiene los siguientes datos: un identificador, existen tres tipos diferentes de requerimientos, por lo que el identificador comienza alguna de las letras *F* (*feature*, para especificar características de manera textual), *G* (*goal*, para establecer los objetivos o alcance deseado) o *R* (para requerimientos ya puntuales), esto seguido de un entero o número decimal no necesariamente consecutivo para ordenarlos; una descripción lo suficientemente detallada para explicar el requerimiento; y una condición que permite saber si el requerimiento pertenecerá o no a un producto.

A partir de aquí, los insumos de especificación de requerimientos, diseño y pruebas, están

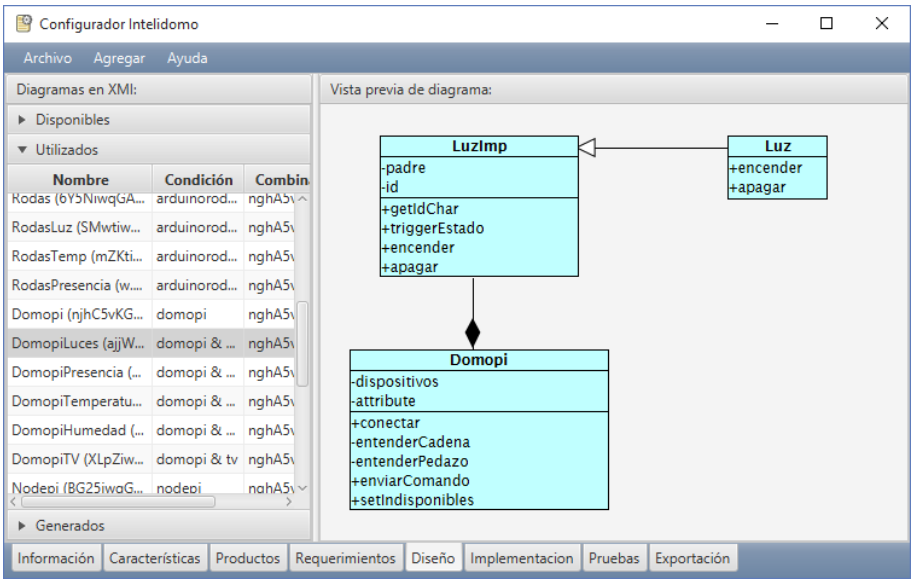

Figura 4.21: Pestaña de diseño de la LPS

condicionados por una expresión booleana basada en las características de la LPS. Los operadores válidos para la condición son & (y, *AND*) y | (o, *OR*). Por ejemplo, para las características definidas en el listado 4.3, una condición válida es *luz & domopi*, o también *domopi | nodepi*. De igual manera se permite la anidación de expresiones con pares de paréntesis, por lo que *nodepi & (luz | presencia)* es una expresión válida. Independientemente de si la característica está originalmente en mayúsculas o minúsculas, se recomienda capturar la condición sólo con minúsculas. Este método es similar al utilizado por DeltaJ para condicionar la aplicación de un módulo delta.

La pestaña de diseño (figura 4.21) está dedicada a mostrar, utilizar y combinar diagramas de clases que representan la estructura de la arquitectura de referencia y cómo esta es modificada para dar vida a diferentes productos o especializaciones. El configurador no tiene la capacidad por sí sólo de crear los diagramas de clases, se utilizó la herramienta *Visual Paradigm 12.2* para elaborar los diagramas y después el proyecto completo se exporta en la versión 2.1 del formato XMI. Al importar tal archivo XMI al configurador, este es capaz de interpretar las instrucciones de representación de cada diagrama definido en el XMI y dibujarlo en pantalla por medio de componentes básicos de JavaFX como etiquetas de texto, contenedores, líneas y polígonos. No se

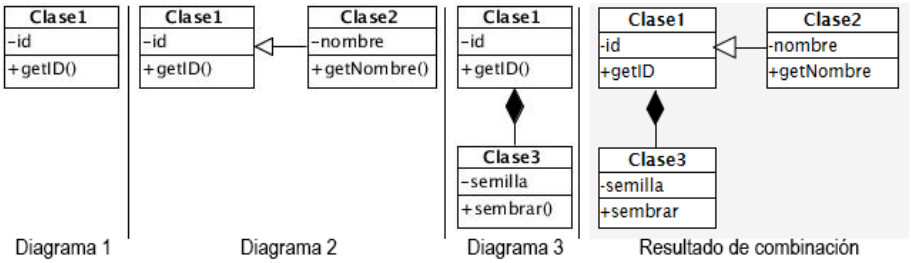

Figura 4.22: Ejemplo de combinación de diagramas de clases

logran percibir diferencias significativas entre la vista de los diagramas en la herramienta *Visual Paradigm 12.2* y el configurador.

Al crear un nuevo proyecto y no importar todavía un XMI, la pestaña permanece bloqueada. Al importar el XMI, los diagramas que contienen se muestran en forma de lista en el contenedor titulado *Diagramas disponibles*. El usuario es capaz de ver cada diagrama al hacer clic en un elemento de la lista y este se dibujará a la derecha de la pantalla. Al hacer clic derecho sobre un elemento de la lista, se muestra un menú contextual con dos opciones: la primera, *utilizar como diagrama base*, establece que el diagrama seleccionado será utilizado como arquitectura de referencia y de él se derivan nuevos diagramas al combinarse con otros; la segunda opción, *utilizar para composición*, establece que el diagrama es secundario y se combinará a otro, para esto, el configurador pregunta al usuario a través de un diálogo la condición para combinar el diagrama y el diagrama objetivo base con el que es capaz de combinarse. Cuando los diagramas se utilizan, desaparecen de la lista del contenedor de *disponibles* y se pasan a la tabla ubicada en el contenedor titulado *utilizados*, mostrando el detalle de si se utilizan como diagrama principal o secundario, su condición, nombre e identificador.

En la figura 4.22 se muestra un ejemplo de combinación de diagramas. El diagrama uno se considera como diagrama base o principal, en este ejemplo es sólo una clase. Los diagramas dos y tres se consideran como secundarios, tienen la capacidad de combinarse con el diagrama uno. Entre los diagramas principal y secundario siempre existe un elemento en común, a partir de tal elemento es como se *pega* un diagrama con otro. El diagrama final tiene todas las clases de los diagramas, así es como se vería reflejada la especialización de una arquitectura de referencia.

El último contenedor con la etiqueta *generados*, tiene una tabla en la cual se van agregando

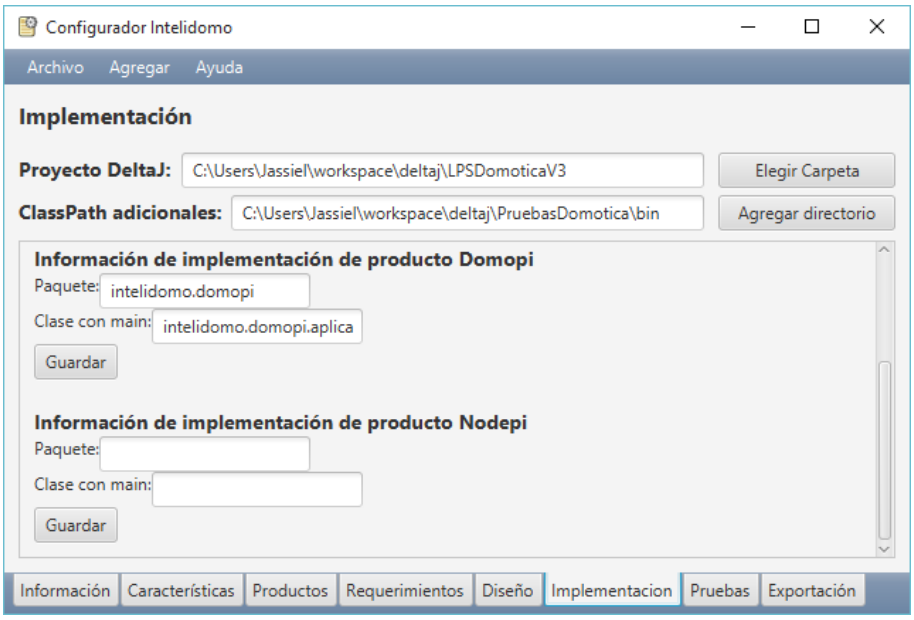

Figura 4.23: Pestaña de implementación de la LPS

los diferentes diagramas producto de una exportación de documentación, indicando el producto al cual corresponde.

La sexta pestaña está dedicada a la implementación. Como se ha dicho, la implementación es hecha totalmente en la parte de DeltaJ, sin embargo, es necesario que el configurador conozca el directorio del proyecto de DeltaJ (tal como se ve en la figura 4.23) para agregar al *classpath* los binarios o archivos compilados del proyecto, así como las bibliotecas *jar* que encuentra (más adelante se verá el porqué es necesario esto). De igual manera, el usuario es capaz de agregar directorios adicionales que contengan clases compiladas o bibliotecas *jar*, o especificar el paquete exclusivo para cada producto y la clase que contiene el método *main* para ejecutar la implementación del producto.

La siguiente pestaña corresponde a las pruebas (figura 4.24, en esta parte se permite capturar la información correspondiente al plan de pruebas. Al dar presionar el botón con la etiqueta *nuevo*, se abre un formulario a la derecha que pide la siguiente información: identificador, consta de un número entero o decimal único para cada prueba, además se utiliza para definir el orden en el que se exporta la información; condición, regla que determina si una prueba pertenece al dominio o a una característica específica; descripción, texto que detalla las acciones a realizar

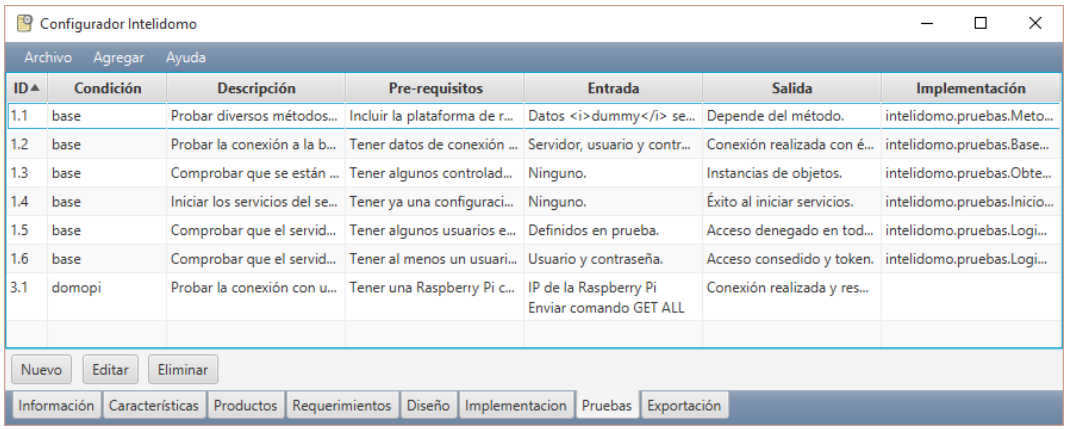

Figura 4.24: Pestaña de pruebas de la LPS

Figura 4.25: Pestaña de exportación

por la prueba y el objetivo final de esta; pre-requisitos, antecedentes o configuraciones a cumplir antes de que la prueba se ejecute para asegurar su correcto funcionamiento; datos de entrada, referencias que la prueba toma y utiliza durante su ejecución; salida esperada, resultado final esperado por la prueba después de ejecutarse; clase de implementación, en esta parte se introduce el nombre canónico de una clase compilada de Java, posteriormente el configurador tratará de ejecutar el método *main* de dicha clase para validar la prueba.

La última pestaña es la correspondiente a la exportación (figura 4.25) en ella se muestran las opciones disponibles para configurar la exportación e iniciar el proceso. Es posible elegir

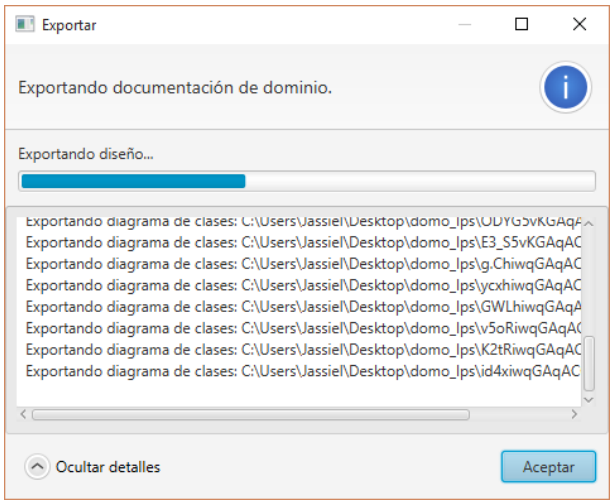

Figura 4.26: Ventana de estado de exportación

entre exportar la documentación de dominio o exportar la documentación específica para un producto. También está la opción de elegir puntualmente que fases del proceso de ingeniería tomar en cuenta para la exportación, elegir un formato (HTML o PDF, siendo HTML el formato por defecto) o cambiar la carpeta donde se almacenará todos los documentos generados por el configurador.

Al presionar el botón con la etiqueta *exportar*, se inicia el proceso de exportación, abriendo una nueva ventana en la que se informa el progreso y las tareas que se realizan al momento, como se ve en la figura 4.26.

Al exportar el documento correspondiente a la ingeniería de dominio, el configurador comienza agregando el título del proyecto y un pequeño resumen acerca de la ingeniería de dominio, seguido por la introducción capturada en la pestaña de *información*. Lo siguiente que hace es agregar una pequeña explicación de lo que es el árbol de características CVL para la LPS, seguida de la imagen con el diagrama CVL correspondiente (para exportar imágenes, JavaFX da la facilidad de hacer *captura de pantalla* de un componente de la interfaz siempre y cuando la ventana y el componente sean visibles para considerar al componente renderizable, de lo contrario la captura sale en blanco, el configurador aprovecha esto utilizando los mismos componentes de la interfaz principal cambiando los modelos a dibujar según convenga, obteniendo el mapa de bits y exportándolo en una ubicación de disco duro). Si en *exportación* se marcó la casilla

referente a exportación de requerimientos, el configurador agrupa todos los requerimientos por condición y después por tipo, y ordenados por su identificador los agrega uno a uno al documento. Si en *exportación* se marcó la casilla referente a exportación de diseño, el configurador dibuja uno a uno los diagramas del XMI marcados como utilizados, obtiene la imagen correspondiente y los agrega uno a uno en el documento indicando su nombre e identificador en el XMI, si los diagramas son condicionados, adicionalmente se indica su condición y el identificador del diagrama con el que se combinan. Si se marcó la casilla de exportación de implementación, se agrega al documento un texto indicando la ruta del proyecto de DeltaJ y los directorios de clases adicionales, también el paquete y punto de ejecución indicado para cada producto. Si se marcó la casilla de pruebas, el configurador las agrupa por condición y las agrega al documento ordenadas por identificador. En el archivo anexo *lps_domótica_dominio.html* se aprecia la documentación de ingeniería de dominio exportada por el configurador para la LPS de domótica.

```
1  import java.util.Random;
2  import java.util.logging.Level;
3  import java.util.logging.Logger;
4  public class PruebaDePrueba {
5    static Logger logger = Logger.getLogger("ConfiguradorTest");
6    static int count = 1;
7    public static void main(String[] args) throws Exception{
8      if(args.length!=3) throw new Exception("Sin argumentos");
9      logger.log(Level.INFO, "Iniciando prueba #{0}", count);
10     int x = new Random().nextInt(34);
11     if(x>=10){
12       logger.log(Level.INFO, "Todo bien");
13       args[2] = "success";
14     }else{
15       logger.log(Level.INFO, "X es menor a 10");
16       args[2] = "failure";
17     }
18   }
19 }
```

Listado 4.2: Ejemplo de implementación de una prueba

Al exportar un documento correspondiente a la ingeniería de aplicación, para cualquier producto, el configurador agrega al documento un pequeño resumen acerca de la ingeniería de aplicación, seguida de la introducción del proyecto. Lo siguiente a agregar es el árbol de

características CVL específico para el producto exportado. Si se marcó la casilla de exportación de requerimientos, el configurador agrupa por tipo y escribe ordenadamente (según el identificador) cada requerimiento cuya condición sea válida para el producto seleccionado. Si se marcó la casilla de exportación de diseño, el configurador comienza por realizar copias de los diagramas base (o de arquitectura de referencia) y después inicia el proceso de combinación de diagramas con aquellos donde su condición cumpla con el producto elegido. Los diagramas resultados de esta combinación, se dibujan en la interfaz para facilitar su exportación, y se agregan al documento. Si se marcó la casilla de exportación de pruebas, el configurador agrega en una tabla toda la información capturada de cada prueba cuya condición concuerde con el producto elegido. A diferencia de la ingeniería de dominio, en la ingeniería de aplicación ya es posible ejecutar las pruebas ya que se tiene por fin un producto implementado. Las pruebas cuya condición concuerde con el producto elegido y tengan una clase válida de implementación, se ejecutan y el detalle de la ejecución se agrega a una tabla y también se indica si el producto pasó la prueba con éxito o no.

Para que el configurador logre capturar la bitácora de la ejecución de una prueba y el resultado final de la prueba, las pruebas se deben ajustar a ciertos lineamientos no invasivos para su ejecución. En el código del listado 4.2 se muestra un ejemplo básico de una prueba que se ajusta a tales lineamientos. La prueba reporta sus operaciones o cualquier otra información a través de la infraestructura *Logger* de Java, el configurador agrega un *listener* al mismo *Logger* ligado a la cadena *ConfiguradorTest*, por lo que cuando la prueba reporta algo en esa misma bitácora, el configurador es capaz capturar el texto y agregarlo al documento. Para permitirle a la prueba indicarle al configurador si el producto pasó la prueba, el configurador envía un arreglo de tres cadenas a la prueba a través del argumento *args[]* del método *main*. El contenido de la primera cadena es el paquete del producto que se está probando, la segunda cadena se refiera al nombre canónico de la clase que inicia al producto, y la tercera cadena se utiliza para reportar el éxito o fallo de la prueba. Así, modificando la tercera cadena, la prueba reporta su resultado, asignando la cadena *success* si hubo éxito, o *failure* en caso contrario. Al terminar la ejecución de la prueba, el configurador revisa el mismo arreglo y la tercera cadena para determinar si se pasó la prueba o no.

Es necesario recordar que el configurador es capaz de ejecutar las pruebas si estas ya están compiladas y se encuentran en el directorio del proyecto de DeltaJ, o cualquier otro que se especifique en los *classpath* adicionales (pestaña *implementación*). Es posible editar la prueba las veces que sean necesarias, esto fuera del entorno del configurador. Cuando el configurador trata ejecutar una prueba, Java busca la clase en los directorios del *classpath* y si la encuentra carga la clase, si la prueba se actualiza en este periodo de tiempo, los cambios no surtirán efecto hasta cerrar y volver a abrir el configurador.

Al terminar todo esto, el documento completo se escribe en disco y se completa la exportación de la documentación del producto elegido, completando los procesos de ingeniería de dominio e ingeniería de aplicación del marco de trabajo utilizado.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<lps nombre="Domotica">
 <Informacion nombreLargo="LPS para domotica" introduccion="Bla..."/>
 <Caracteristicas cvl="arbol.cvl">
  <Caracteristica>Luz</Caracteristica>
  <Caracteristica>Presencia</Caracteristica>
  <Caracteristica>Temperatura</Caracteristica>
  <Caracteristica>ArduinoRodas</Caracteristica>
  <Caracteristica>Domopi</Caracteristica>
  <Caracteristica>Nodepi</Caracteristica>
 </Caracteristicas>
 <Requerimientos>
  <Requerimiento condicion="base" id="G1.1"
  descripcion="Ofrecer un sistema de control..."/>
  <Requerimiento condicion="luz" id="F2"
  descripcion="Control de luces."/>
  <Requerimiento condicion="nodepi &amp;amp; luces" id="R41.1"
  descripcion="Un dispositivo MCU se..."/>
 </Requerimientos>
 <Disenio>
  <Diagrama id="nghA5vKGAqACQAmA" condicion="base"/>
  <Diagrama id="E3_S5vKGAqACQAXq" condicion="luz"
  combinableCon="nghA5vKGAqACQAmA"/>
 </Disenio>
 <Pruebas>
  <Prueba id="1.1" condicion="base" descripcion="..." requisitos="..."
  entrada="..." salida="..." implementacion="..."/>
  <Prueba id="1.2" condicion="base" descripcion="..." entrada="..."
  salida="..." implementacion="..."/>
 </Pruebas>
 <Productos>
  <Producto nombre="Domopi" paquete="domopi"
  clasePrincipal="domopi.Inicio">
   <Caracteristica>Luz</Caracteristica>
   <Caracteristica>Presencia</Caracteristica>
   <Caracteristica>DomoPi</Caracteristica>
  </Producto>
 </Productos>
</lps>
```

Listado 4.3: Versión muy reducida de un ejemplo del XML de un proyecto del configurador

```
1  SPL intelidomo {
2   Features={base, pgsql, mysql, voz,luces, presencia, temperatura,
     ↪  humedad, tv, cortina, alarma, fuego, gases, rodas, domopi, nodemcu,
     ↪  milight}
3   Deltas={dBase, dDatabase, dModelos, dRules, dServer, dPgsql, dLuces,
     ↪  dPresencia, dTemperatura, dHumedad, dTV, dCortina, dAlarma,
     ↪  dLuzRGB, dRodas, dRodasLuces, dRodasPresencia, dRodasTemperatura,
     ↪  dDomopi, dDomopiLuces, dDomopiPresencia, dDomopiTemperatura,
     ↪  dDomopiHumedad, dDomopiTV, dNodepi, dNodepiLucesDim,
     ↪  dNodepiTemperatura, dNodepiHumedad, dNodepiCortina, dNodepiFuego,
     ↪  dNodepiGas, dVoz, dVozLuces, dVozTemperatura, dVozHumd, dVozTV,
     ↪  dMilight}
4   Constraints {mysql^pgsql; alarma&fuego; alarma&gases;}
5   Partitions { {dBase,dDatabase,dModelos,dServer} when (base); {dPgsql}
     ↪  when (pgsql); {dLuces} when (luces); {dPresencia} when (presencia);
     ↪  {dTemperatura} when (temperatura); {dHumedad} when (humedad);
     ↪  {dCortina} when (cortina); {dTV} when (tv); {dAlarma} when
     ↪  (alarma); {dLuzRGB} when (milight); {dRodas} when (rodas);
     ↪  {dRodasLuces} when (rodas & luces); {dRodasPresencia} when (rodas &
     ↪  presencia); {dRodasTemperatura} when (rodas & temperatura);
     ↪  {dDomopi} when (domopi); {dDomopiLuces} when (domopi & luces);
     ↪  {dDomopiPresencia} when (domopi & presencia); {dDomopiTemperatura}
     ↪  when (domopi & temperatura); {dDomopiHumedad} when (domopi &
     ↪  humedad); {dDomopiTV} when (domopi & tv); {dNodepi} when (nodemcu);
     ↪  {dNodepiLucesDim} when (nodemcu & luces); {dNodepiTemperatura} when
     ↪  (nodemcu & temperatura); {dNodepiHumedad} when (nodemcu & humedad);
     ↪  {dNodepiCortina} when (nodemcu & cortina); {dNodepiFuego} when
     ↪  (nodemcu & fuego); {dNodepiGas} when (nodemcu & gases); {dMilight}
     ↪  when (milight); {dVoz} when (voz); {dVozLuces} when (voz & luces);
     ↪  {dVozHumd} when (voz & humedad); {dVozTemperatura} when (voz &
     ↪  temperatura); {dVozTV} when (voz & tv);}
6   Products {
7    Rodas={base,pgsql, luces, presencia, temperatura, rodas};
8    Domopi={base, pgsql, luces, presencia, temperatura, humedad, tv,
      ↪  domopi};
9    Nodepi={base,pgsql, luces, temperatura, humedad, cortina, alarma,
      ↪  fuego, gases,nodemcu};
10   Completo={base,pgsql, luces, presencia, temperatura, humedad, tv,
      ↪  cortina, alarma, fuego, gases, rodas, domopi, nodemcu};
11   Plus={base,pgsql, luces, presencia, temperatura, humedad, tv, cortina,
      ↪  alarma, fuego, gases, rodas, domopi, nodemcu, milight, voz};
12  }
13 }
```

Listado 4.4: Archivo de configuración de DeltaJ

4.2.2. Integración con DeltaJ

Uno de los objetivos del configurador es reducir el tiempo invertido en generar la documentación específica para un producto, el proceso seguido por el configurador imita el esquema composicional utilizado en DeltaJ. El XML del proyecto generado por el configurador tiene algunas similitudes con el archivo de configuración de DeltaJ. En el listado 4.4 se muestra el archivo de configuración completo realizado para la LPS de domótica.

El XML del listado 4.3 está incompleto en contenido pero contiene la misma estructura que el archivo original. Comparando el listado 4.4 con el listado 4.3 se deduce que: en el XML las características se definen en etiquetas *Característica* (líneas 4 a 11) mientras que en DeltaJ esto se consigue a través de cadenas literales separadas por comas (línea 2); en DeltaJ los deltas se definen de igual manera con literales (línea 3) y se crea un archivo con extensión *.deltaj* por cada uno, donde residen las instrucciones para el código fuente, en el configurador no se definen deltas pero el concepto está implícito ya que se toma en cuenta una serie de insumos de referencia (los requerimientos, diagramas y pruebas) y a estos se les agregan los específicos para cada producto al generar su documentación; (3) DeltaJ permite especificar restricciones *alternativas*, *incluyentes* o *excluyentes* para las características a través de operadores OR, AND y XOR, en el configurador estas restricciones se definen por el diagrama CVL y no se reflejan en el XML del proyecto (pero se hace una referencia al archivo CVL en la línea 4); DeltaJ permite definir cuándo se aplican los deltas indicando una condición de características y el nombre del delta (línea 5), en el configurador se aplican condiciones equivalentes para cada artefacto y se aprecia en el XML del proyecto el atributo *condicion* en etiquetas referentes a requerimientos, diagramas y pruebas (líneas 12 a 30); DeltaJ permite especificar los productos a través de literales y llaves que encierran las características que tiene cada uno (líneas 6 a 12), mientras que en el XML del configurador esto se aprecia de manera similar con etiquetas *Producto* y *Caracteristica* (líneas 32 a 37).

Al solicitar la exportación de los productos en DeltaJ, este permite elegir qué productos generar a través de un cuadro de diálogo (figura 4.27). Se marcan las productos deseados y al dar clic al botón *Continue* se comienza la generación del código fuente de los productos elegidos. En

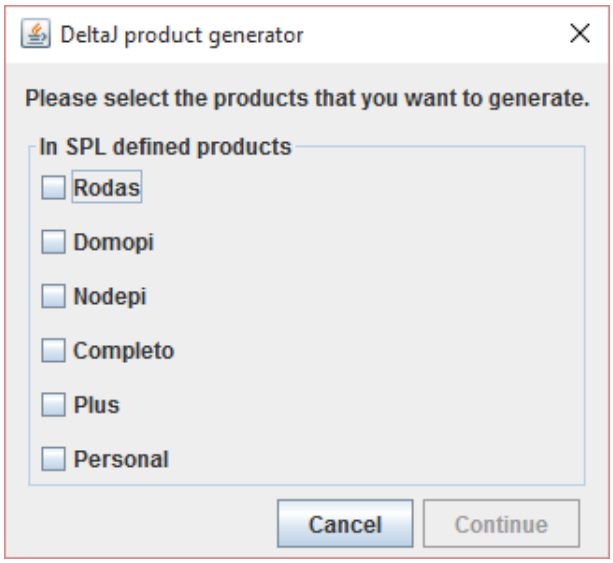

Figura 4.27: Exportador de DeltaJ

el configurador se genera sólo un producto a la vez, tal como se muestra en la figura 4.28.

El configurador desarrollado no compite con DeltaJ, es una herramienta que complementa las funcionalidades de DeltaJ y ayuda a flexibilizar aún más el desarrollo de Líneas de Productos de Software y completando el ciclo del marco de trabajo utilizado.

La automatización de pruebas es también una virtud del configurador y por ello hace referencia a la carpeta del proyecto DeltaJ para cargar sus clases y facilitar la ejecución de las pruebas.

4.3. Controladores de ambiente domótico desarrollados

Para la aplicación del caso de estudio y el desarrollo de la línea, se consideró desarrollar una serie de dispositivos que actuaran como los sensores y actuadores básicos de un ambiente domótico, evitando la adquisición de dispositivos existentes en el mercado que no cubren todas las necesidades o se desvían del objetivo del proyecto y tienen costos elevados, de esta manera se logra ofrecer un producto más llamativo al competir en precio con otras alternativas de domótica en el mercado.

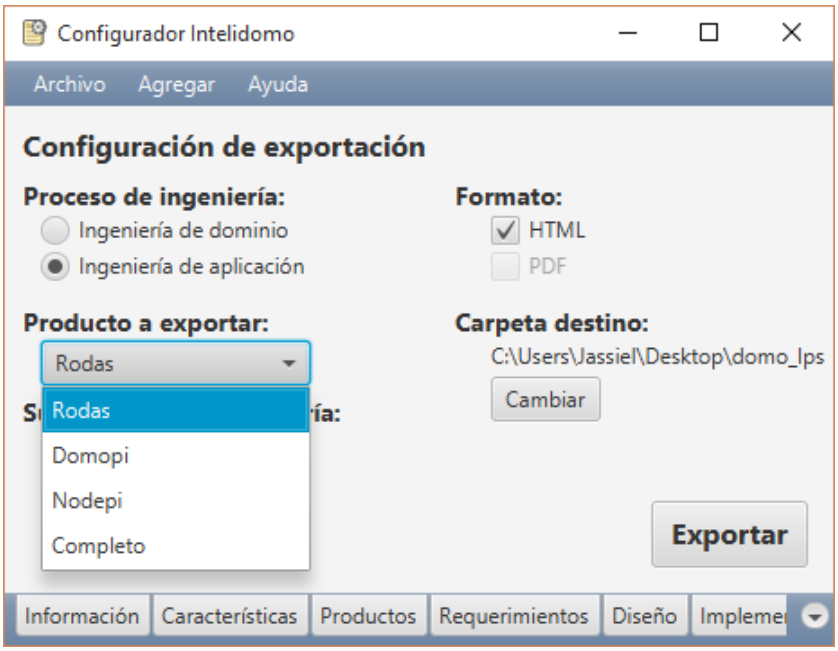

Figura 4.28: Eligiendo un producto a exportar en el configurador

Para el desarrollo de los dispositivos se emplearon algunos de los microcontroladores más populares, ya sea porque facilitan la aplicación del concepto del Internet de las Cosas (IoT, *Internet of Things*), o porque cuentan con buena cantidad de documentación y una gran comunidad de respaldo.

4.3.1. Domopi

Se nombró *Domopi* al conjunto de software desarrollado y hardware utilizado para dar vida a ciertas características de control domótico (control de luces, lectura de temperatura, lectura de humedad, lectura de presencia y control de televisiones por infrarrojo), esto, basándose sobre el dispositivo *Raspberry Pi*.

Raspberry Pi [22] es una pequeña computadora de bajo costo y totalmente funcional del tamaño de una tarjeta de crédito, tiene varios puertos como una salida de vídeo HDMI, cuatro puertos USB, un puerto Ethernet y 40 puertos GPIO (el número de puertos USB y GPIO varía según la versión). Soporta diferentes sistemas operativos, la mayoría basados en Linux, es posible conectarle un teclado, un ratón y un monitor y utilizarla como una computadora de escritorio

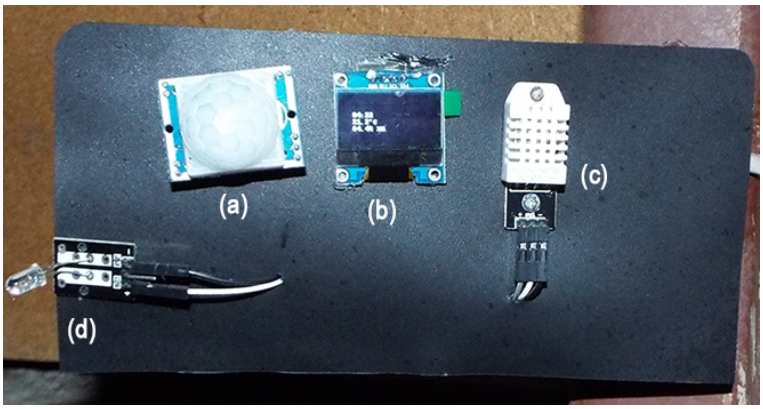

Figura 4.29: Sensores empotrados y conectados a la Raspberry Pi

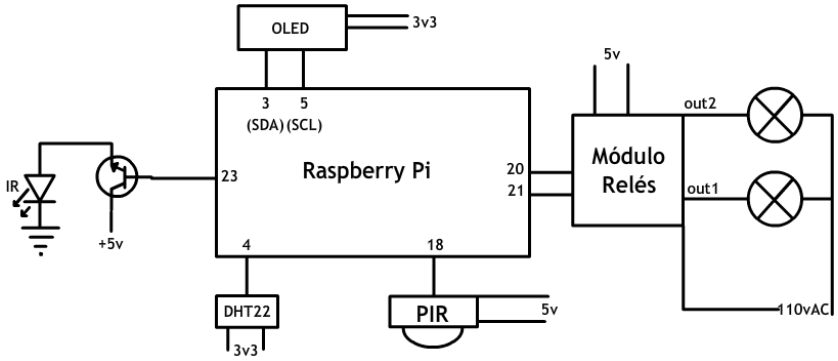

Figura 4.30: Diagrama de conexión de sensores a la Raspberry Pi

o portátil cualquiera. La Raspberry Pi versión 2B se diferencia de sus versiones anteriores por albergar un procesador ARM Cortex-A7 900MHz de cuatro núcleos y tener capacidad de 1Gb de RAM. La Rasbperry Pi tiene la habilidad de interactuar con el mundo exterior gracias a sus puertos GPIO, ha sido utilizada en un sin fin de proyectos digitales DIY (*do it yourself*), desde cajas musicales hasta estaciones climatológicas o sistemas de monitoreo infrarrojo [23].

Las características implementadas sobre la Raspberry Pi, son las de control de luces, lectura de temperatura y humedad, lectura de presencia y control de televisiones. En la figura 4.29 se aprecia (a) un sensor de presencia, (b) una pequeña pantalla OLED para mostrar información de los sensores, (c) un sensor de humedad y temperatura DHT22; y (d) un diodo emisor de infrarrojos apuntando hacia una televisión. Todos estos módulos se conectan a la Raspberry Pi a través de un cable UTP. En la figura 4.30 se aprecia el modo de conexión de los módulos con la

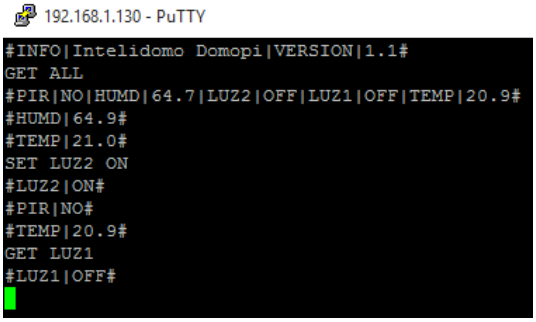

Figura 4.31: *Charlando* con la Raspberry Pi

Raspberry Pi.

Se implementó un *script* en Python para la Raspberry Pi, encargado de leer el estado del sensor de presencia en tiempo real, obtener la temperatura y humedad del sensor cada 30 segundos, así como mostrar información actualizada de los sensores en la pantalla OLED. El *script* también provee un servidor TCP/IP que permite a las aplicaciones interactuar con los actuadores y conocer el estado de los sensores. En la figura 4.31 se abre una sesión con el servidor. Cuando el servidor acepta la conexión, envía primeramente un mensaje de bienvenida que indica la versión y el nombre del servicio. Cuando el cliente envía la cadena *GET ALL*, el servidor devuelve el estado de todos los dispositivos disponibles. Las peticiones siempre comienzan con GET o SET, mientras que las respuestas siempre se conforman de pares, indicando la clave del dispositivo y su valor.

Mientras el cliente siga conectado, el servidor continuará enviando información actualizada del estado de los dispositivos sin que el cliente los pida. El cliente tiene la posibilidad de seguir enviando peticiones durante la sesión. El servidor Intelidomo interactúa con la Raspberry Pi de este modo, gracias a que la conexión permanece abierta, éste se entera en tiempo real del estado de los dispositivos disponibles, sin necesidad de estar enviando peticiones frecuentemente para actualizar su información.

Para controlar dispositivos que disponen de interfaz de infrarrojos, se hace uso de un diodo emisor de infrarrojos conectado a la Raspberry Pi (figura 4.29) e instalar LIRC, una biblioteca para sistemas Linux que facilita la recepción y envío de señales infrarrojas. El diodo infrarrojo debe estar siempre apuntando hacia el dispositivo a controlar. Para configurar LIRC, es necesario buscar en la página oficial el listado de controles remotos conocidos, elegir el más adecuado

según el dispositivo a controlar y descargar la configuración. En el caso de una televisión LG 55LB7200, la configuración del listado 4.5 es la adecuada.

Después de agregar la configuración, para enviar señales infrarrojas basta con ejecutar un comando desde la terminal de la Raspberry Pi que indique la configuración a usar y el comando a enviar, por ejemplo, ejecutar el comando *irsend SEND_ONCE LG_MKJ40653807 KEY_VOLUMEUP* hará que el volumen de la televisión aumente. Se creó un servidor HTTP en Python que facilita la ejecución y envío de comandos. Al acceder a la dirección *http://ip-raspberry:8540/key/KEY_VOLUMEUP*, se ejecuta el mismo comando anterior para subir el volumen y es posible modificar la dirección para enviar cualquier comando registrado en la configuración.

Utilizando la interfaz del servidor HTTP, el servidor Intelidomo es capaz de controlar la televisión, pero lo mismo aplica a cualquier otro dispositivo que tenga receptor infrarrojo y cuya lista de códigos sea conocida.

4.3.2. Nodepi

Se nombró *Nodepi* al conjunto de software desarrollado y hardware utilizado para dar vida a ciertas características de control domótico (control de luces, lectura de temperatura, lectura de humedad, lectura de presencia de gases peligrosos, control de cortinas), esto, basándose sobre el dispositivo NodeMCU.

NodeMCU es una plataforma de desarrollo de código abierto que facilita la creación de prototipos orientados al Internet de las Cosas con unas pocas líneas de código en el lenguaje Lua. Es de código abierto, interactivo, programable, de bajo costo y viene habilitado para conexiones WiFi. Su API está orientada a aplicaciones de red, lo que facilita a los programadores escribir y ejecutar código en el dispositivo [24]. El kit de desarrollo se basa en el chip ESP8266, integra puertos GPIO, PWM, IIC y un puerto Análogico-Digital en una tarjeta.

El primer dispositivo desarrollado sobre la plataforma de NodeMCU está orientado a las características de temperatura, humedad, alarma de fuego y alarma de gases peligrosos.

En la figura 4.32 se muestra el dispositivo mencionado (antes de ser protegido por una cubierta

```
1 begin remote
2  name LG_MKJ40653807
3  bits          16
4  flags         SPACE_ENC|CONST_LENGTH
5  eps           30
6  aeps          100
7  header        8994 4417
8  one           590 1649
9  zero           590 511
10 ptrail         568
11 pre_data_bits 16
12 pre_data      0x20DF
13 gap           107287
14 toggle_bit_mask 0x0
15   begin codes
16     KEY_POWER        0x10EF
17     KEY_MENU         0xC23D
18     KEY_UP           0x02FD
19     KEY_DOWN         0x827D
20     KEY_LEFT         0xE01F
21     KEY_RIGHT        0x609F
22     KEY_OK           0x22DD
23     KEY_VOLUMEUP     0x40BF
24     KEY_VOLUMEDOWN   0xC03F
25     KEY_MUTE         0x906F
26     KEY_CHANNELUP    0x00FF
27     KEY_CHANNELDOWN  0x807F
28     KEY_1            0x8877
29     KEY_2            0x48B7
30     KEY_3            0xC837
31     KEY_4            0x28D7
32     KEY_5            0xA857
33     KEY_6            0x6897
34     KEY_7            0xE817
35     KEY_8            0x18E7
36     KEY_9            0x9867
37     KEY_0            0x08F7
38   end codes
39 end remote
```

Listado 4.5: Archivo de configuración para LIRC

plástica), las partes de hardware utilizadas se describen a continuación: (a) microcontrolador NodeMCU donde reside la lógica del dispositivo y permite su interacción a través de conexiones TCP/IP, (b) adaptador de alimentación con entrada de 6V 12V y salidas de 3.3V y 5V; (c) sensor

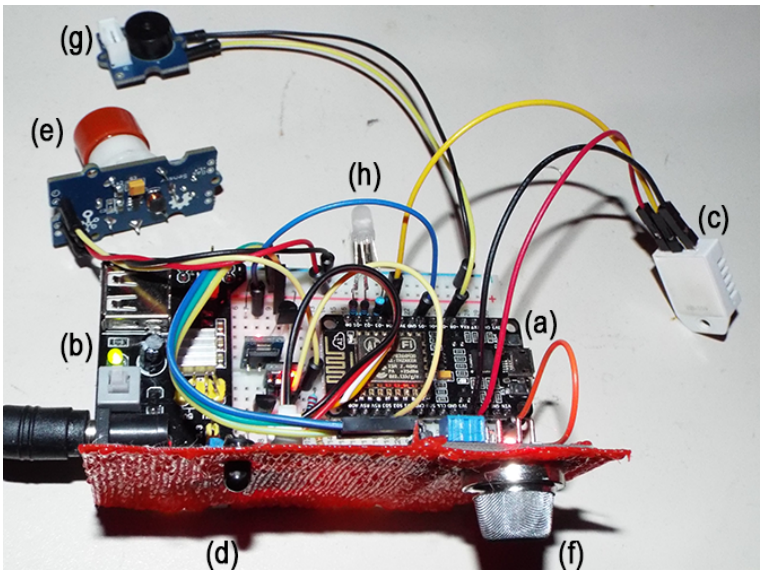

Figura 4.32: Dispositivo de lectura de diversos sensores

de humedad y temperatura DHT22 de alta precisión; (d) sensor de flama, se dispara al detectar flamas a distancias no superiores a 2 metros, a veces también se dispara al detectar luz fuerte indirecta del sol; (e) sensor de gas MQ-9, es capaz de detectar gas LP, metano, vapores de alcohol y monóxido de carbono, el voltaje de salida varía de 0 a 5v según las partes por millón detectadas de alguno de los gases mencionados; (f) sensor de gas MQ-2, es capaz de detectar monóxido de carbono, vapores de alcohol, propano, metano, gas LP, butano, humo e hidrógeno; (g) zumbador y (h) LED RGB. Existen muchos tipos de sensores de gases, sin embargo la sensibilidad y rango de detección varían de sensor a sensor y de gas a gas, la información completa acerca de cual sensor escoger según una necesidad específica se encuentra disponible en [25]. Se ha utilizado más de un sensor para mejorar la respuesta y prevención ante gases peligrosos.

En la figura 4.33 se muestra el diagrama de conexión, algunos de los módulos utilizados (flama y los dos de gases) trabajan sobre 5v, sin embargo el microcontrolador NodeMCU es un dispositivo de 3.3v, por lo que se requirió de divisores de voltaje, reguladores de voltaje, algunos transistores y resistencias para evitar dañar al microcontrolador.

En el código de los listados 4.6 y 4.7 se muestra la lógica utilizada para el microcontrolador NodeMCU, básicamente los pasos son: (1) definir los datos de conexión con el punto de acceso; (2) declarar las variables de humedad y temperatura, definir un temporizador que cada 30 segundos

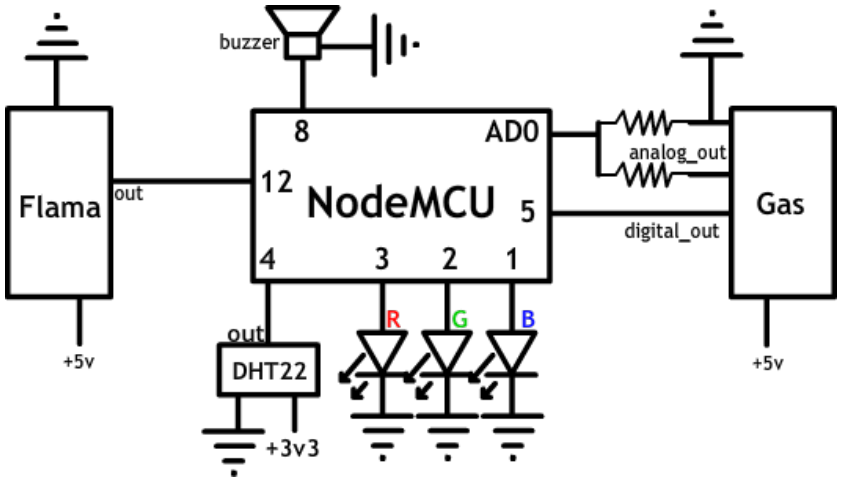

Figura 4.33: Diagrama de conexión con sensores

realice la lectura del sensor DHT22 y actualizar las variables; (3) definir la función del zumbido controlando las veces que suena y el tiempo entre zumbido; (4) definir un temporizador que cada 100 milisegundos realice la lectura del sensor de gas analógico y si el valor obtenido sobrepasa cierto punto, llamar a la función de zumbido para alertar a los usuarios; (5) definir un temporizador que cada 100 milisegundos realice la lectura del sensor de gas digital y si el valor obtenido es diferente de 1, llamar a la función de zumbido para alertar a los usuarios y actualizar la variable de gas; (6) definir un temporizador que cada 100 milisegundos realice la lectura del sensor de flama y si el resultado es diferente de 1, llamar la función de zumbido y actualizar la variable de flama; y (7) definir un servidor TCP/IP sobre el puerto 80 que al recibir la cadena *get values* desde un cliente, envíe una cadena con todos los valores resultado de la lectura de los sensores en el formato *#key1|value1|keyn|valuen#*.

Cuando el servidor domótico encuentra en la base de datos un dispositivo llamado *Nodepi-Sensors*, carga el controlador específico, el cual manda el comando *get values* cada segundo al dispositivo NodeMCU para recuperar los valores de los sensores y de esta manera implementar al menos las características de temperatura, humedad, alarma de fuego y alarma de gases peligrosos.

El segundo dispositivo desarrollado sobre la plataforma de NodeMCU está orientado a la característica de control de cortinas. En la figura 4.34 se muestra el dispositivo mencionado. Está conformado por las siguientes partes de hardware: (a) microcontrolador NodeMCU; (b) fuente

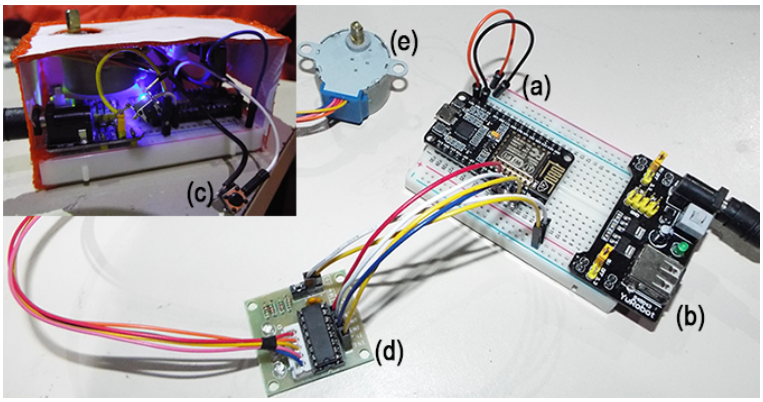

Figura 4.34: Dispositivo de control de cortinas

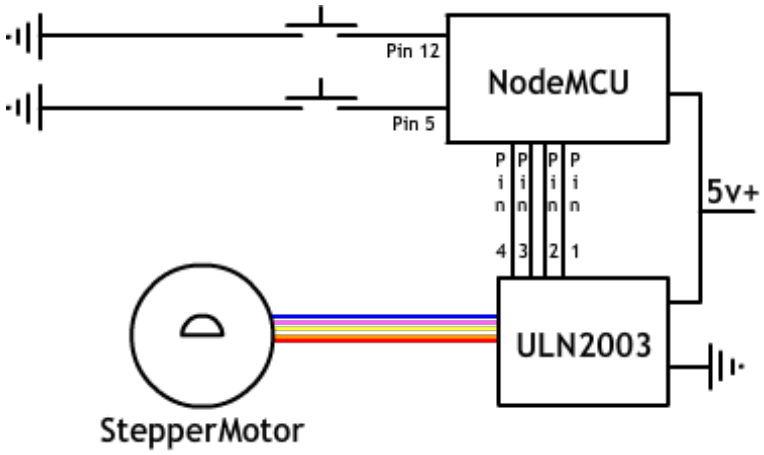

Figura 4.35: Diagrama de conexión del dispositivo de cortina

de poder; (c) botones para controlar manualmente la cortina; (d) módulo controlador de motor paso por paso con microchip ULN2003; y (e) motor de paso por paso de 5V de 4 fases. El motor se considera de precisión, a través del microcontrolador se decide cuántos pasos exactamente avanzar o retroceder. Su movimiento es lento, tarda aproximadamente 3 segundos y medio para dar una vuelta completa, la cual consta de 512 pasos. El engranaje del motor es robusto y con un buen *torque*, no es posible girar el pivote del motor con la fuerza de la mano por lo que permanece en su posición incluso cuando está apagado.

En la figura 4.35 se muestra el diagrama de conexión del microcontrolador con las demás partes de hardware. En los códigos de los listados 4.8 y 4.9 se muestra la lógica del dispositivo, se requiere de 4 salidas activadas en un orden específico (véase el método *forward* de la línea 9

del listado 4.8) para realizar un paso en el motor. El control de la cortina se basa en porcentajes, siendo 0 % una cortina totalmente abierta y 100 % una cortina totalmente cerrada, por lo que se define el número máximo de pasos para considerarse como 100 % y que el motor no gire de más. El servidor del dispositivo acepta una serie de comandos, al recibir el comando *set position* ***50***, se tomará el número *50* y se pasará como argumento a la función *goto*, así la cortina se moverá a la mitad de su posición máxima, estando en un estado de medio abierta o medio cerrada.

Otros comandos que el dispositivo acepta son: *get position* permite recuperar la posición actual de la cortina, es utilizada cada 5 minutos por el servidor domótico para conocer la posición de la cortina en caso de que haya sido controlada manualmente o cambiada por otro cliente conectado (ejemplo de respuesta: *#position|50#*); *get maxsteps* permite conocer el número máximo de pasos que al motor se le permite dar (ejemplo de respuesta: *#maxsteps|1500#*); *set maxsteps* ***1200*** permite cambiar el número máximo de pasos que se le permite al motor dar, útil para configurar un dispositivo a diferentes tamaños de cortina, en este caso el número máximo de pasos se establecería como 1200; *get currentsteps* permite conocer la posición actual de la cortina en pasos del motor (ejemplo de respuesta: *#steps|150#*); *go forward* ***20*** mueve 20 pasos hacia adelante el motor; *go backwards* ***50*** mueve 50 pasos el motor hacia atrás; *set as initial* establece la posición actual como la inicial o 0 %; *set as maximum* establece la posición actual como la máxima o 100 %; y *save config* permite guardar en memoria ROM la posición actual y posición máxima para recuperar los valores en caso de interrupción o desconexión de la energía eléctrica, esto para evitar reprogramar el dispositivo cada que se adapte a una nueva cortina. El dispositivo acepta peticiones HTTP y responde únicamente con un texto genérico. Cualquier otra petición obtendrá como respuesta un error con la siguiente cadena: *#error|invalid command#*.

El motor se montó sobre un cortinero de metal, y como cortina una tela sintética opaca lo suficientemente flexible para bajar sin quedarse doblada, o subir y enrollarse adecuadamente. El hardware se encapsuló con una cubierta plástica a la medida.

El tercer dispositivo desarrollado sobre la plataforma de NodeMCU se elaboró para implementar unas luces de intensidad regulable. En la figura 4.36 se muestra el dispositivo y el diagrama de conexión. La placa sobre la cual se monta el microcontrolador NodeMCU es un controlador cuyo objetivo original es controlar motores de corriente directa, sin embargo se ideó utilizarlo

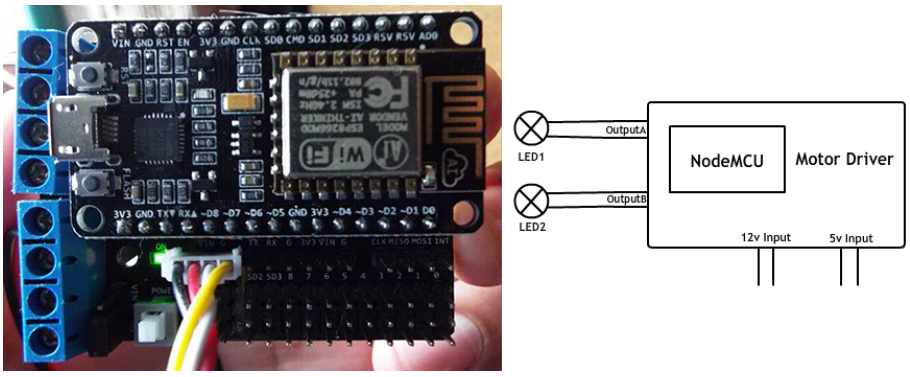

Figura 4.36: Dispositivo de luces regulables

para alimentar luces LED de alta luminosidad y así contar con una característica agregada de luces capaces de regular su brillo y adaptarse a diferentes ambientes o eventos según el estado de humor del usuario.

PWM es una técnica utilizada para modular el ancho de un pulso (por sus siglas en inglés, *pulse-width modulation*) de una señal o fuente de energía, ya sea para transmitir información a través de un canal de comunicaciones o para controlar la cantidad de energía que se envía a una carga [26]. Así, el microcontrolador es capaz de variar la intensidad de una luz cambiando la frecuencia del pulso que envía al controlador de motores. Esto se refleja en el código del listado 4.10, donde además se aprecia un servidor capaz de responder a los comandos *get light1* o *get light2* para obtener la intensidad actual de la primera luz o la segunda, respectivamente; o también los comandos *set light1 to **100*** y *set light2 to **0*** para cambiar la intensidad de las luces, donde **100** equivale a una luz totalmente encendida y **0** a una luz totalmente apagada, cualquier número entero entre 0 y 100 es aceptado.

4.4. Interoperabilidad con otros dispositivos de hardware

Para asegurar la flexibilidad y extensibilidad del diseño de la arquitectura de referencia de la LPS, se incorporó soporte de otros dispositivos diferentes a los desarrollados. El reto consistió en trabajar con dispositivos con protocolos y maneras de funcionar diferentes a las propuestas, y

abstraer correctamente la funcionalidad de los dispositivos sin contaminar o desequilibrar el flujo de trabajo normal del sistema y de los productos generados en la LPS. Al hacer esto, se prueba que la LPS es capaz de generar productos que soporten distintos dispositivos, controladores y protocolos al mismo tiempo, sin crear colisiones o exclusiones entre uno y otro, asegurando una cooperación de distintos tipos de dispositivos no importando el desarrollador (o proveedor), apegándose al concepto de domótica.

4.4.1. Arduino Rodas

La empresa Rodas Computación S.A. de C.V., interesada en la automatización de ciertas áreas de índole hotelera, decidió invertir esfuerzos en el desarrollo de dispositivos que permitieran cubrir tales necesidades. Se basó sobre la plataforma Arduino para desarrollar sus prototipos.

Arduino es una placa de desarrollo simple y robusta, es una de las opciones disponibles más sencillas para crear dispositivos confiables programados para el mundo real [27].

En la primera versión del prototipo, la empresa implementó las características de control de luces, detección de presencia, censado de temperatura, control de ventilación y detección de incendios. Sin embargo, al momento de la integración del dispositivo al sistema Intelidomo, el prototipo o dispositivo físico real sólo contaba con las características de luz, presencia y temperatura, por lo que estas fueron las únicas implementadas en la LPS en ese momento. Para la empresa no fue posible prestar sus prototipos o elaborar más de uno, por lo que la empresa desarrolló también una aplicación que simulara las funciones, operaciones y protocolo de comunicación del dispositivo. Se utilizó este simulador (figura 4.37) para realizar las primeras pruebas, al implementar las características en las primeras versiones de la LPS.

La comunicación con el dispositivo se logra a través de TCP/IP, enviando cadenas en un formato específico que sirven como comandos. El sistema Intelidomo envía estos comandos (presentes en la tabla 4.2) para reconocer cambios en su estado. Para saber el estado de una luz, el sistema envía el comando requerido cada 5 segundos; para saber saber la temperatura actual, el sistema envía el comando cada 10; y para saber si hay cambios en el estado de presencia, el sistema pregunta el estado cada 3 segundos.

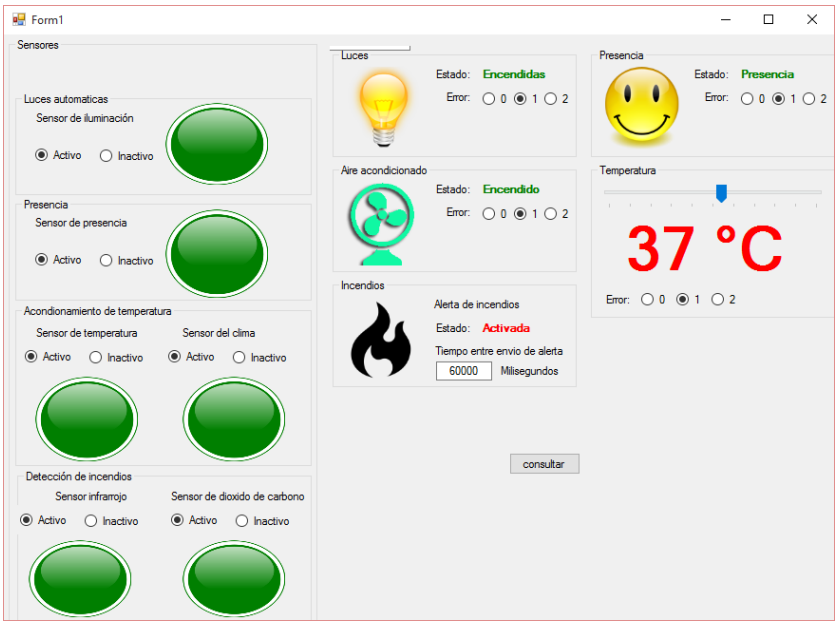

Figura 4.37: Simulador del dispositivo Arduino

Acción	**Comando**	**Respuesta**
Consultar el estado de los sensores	GET /Test	#$ST\|0\|$AC|0|$L\|0\|$P|0#
Consultar la temperatura	GET /Temp	#$ST\|00.00\|$E|0#
Saber si hay presencia	GET /Pre	#$P\|1\|$E|0#
Saber el estado de la luz	GET /Light	#$L\|1\|$E|0#
Encender la luz	GET /Onlight	#$L\|1\|$E|0#
Apagar la luz	GET /Offlight	#$L\|0\|$E|0#

Tabla 4.2: Comandos del dispositivo Arduino

Actualmente la empresa continúa trabajando sobre sus prototipos, han actualizado el protocolo de comunicación de su dispositivo y planean agregar nuevas características. Se contempla agregar también soporte para dichos dispositivos si la empresa decide compartir sus nuevos protocolos.

4.4.2. Luces MiLight

MiLight es una empresa dedicada a manufacturar y distribuir dispositivos de luz con la capacidad de ser controladas a distancia. Ofrecen una amplia variedad, desde focos incandescentes y bulbos fluorescentes hasta luces LED. Sus dispositivos incorporan un controlador embebido

Figura 4.38: Luz RGBW y puente WiFi de MiLight

capaz de recibir señales de radio en la frecuencia de 2.4Ghz, adicionalmente ofrecen *puentes* que operan sobre el estándar WiFi, reenviando los comandos entre un medio y otro para controlar la luz. De igual manera ofrecen gratuitamente aplicaciones para dispositivos Android o iOS.

Se adquirió la luz LED RGBW y el puente WiFi de la figura 4.38. La luz se conecta directamente al suministro eléctrico a través de un socket convencional. El puente se alimenta conectándolo en cualquier puerto USB y es capaz de controlar una luz a la vez. El protocolo de comunicación del dispositivo es cerrado, no hay documentación oficial ofrecida por los proveedores que permitan a terceros crear aplicaciones para controlar sus luces.

Para conocer el protocolo, se instaló en una computadora el programa capturador de paquetes de red WireShark. Se configuró a la computadora como punto de acceso de una red WiFi. Se siguieron las instrucciones del proveedor para emparejar el punte con la luz LED, así como para conectar el puente al punto de acceso. Se descargó la aplicación de control a un teléfono Android y se inició la captura de paquetes.

Después de filtrar los paquetes, se encontró que la comunicación entre la aplicación y el puente no contenía algún tipo de cifrado, por lo que fue relativamente fácil desentrañar el protocolo, realizando acciones específicas en la aplicación y viendo como reaccionaba la luz.

En resumen, el protocolo consta de enviar tres bytes de información por comando a la IP del dispositivo (o a la IP de difusión) sobre el protocolo UDP. Se listan los comandos a continuación:

- **0x21 0x00 0x55:** Apaga la luz.

- **0x22 0x00 0x55:** Enciende la luz.
- **0x23 0x00 0x55:** Aumenta el brillo de la luz.
- **0x24 0x00 0x55:** Reduce el brillo de la luz.
- **0x25 0x00 0x55:** Aumenta la velocidad del efecto actual (si está alguno activo).
- **0x26 0x00 0x55:** Reduce la velocidad del efecto actual (si alguno está activo).
- **0x27 0x00 0x55:** Avanza entre los efectos disponibles en la luz (blanca, color estático, arcoíris, estroboscópica, entre varias opciones disponibles).
- **0x28 0x00 0x55:** Retrocede entre los efectos disponibles en la luz.
- **0x20 [0x00-0xFF] 0x55:** Permite cambiar el color actual que se está mostrando en la luz.

Una vez conociendo el protocolo, se logró incluir tal funcionalidad en el sistema Intelidomo y permitir desde una misma interfaz controlar luces LED RGBW, al menos para este proveedor. Se creó un cliente UDP que envía los comandos sobre la red a petición del usuario y se incluyó en el servidor.

```
wifi.setmode(wifi.STATION)
wifi.sta.config("Infinitum","PSW")
tmp = "na";hmd = "na"
tmr.alarm(0,30000,1,function()
  status,temp,humi,temp_decimial,humi_decimial = dht.read(4)
  if( status == dht.OK ) then
    tmp = math.floor(temp).."."..string.sub(temp_decimial,1,1)
    hmd = math.floor(humi).."."..string.sub(humi_decimial,1,1)
  end
end)
gpio.mode(8,gpio.OUTPUT);zumbing = 0
function beep(times)
  if zumbing==0 then
    b = 0; c = 1; max = times*2+6; zumbing = 1
    tmr.alarm(1,100,1,function()
      x = c%2; c = c + 1
      if b < times then gpio.write(8,x)
      elseif b == times then gpio.write(8,0) end
      if x==1 then b = b + 1 end
      if(c==max) then tmr.stop(1); zumbing = 0 end
    end)
  end
end
gpio.mode(1,gpio.OUTPUT);gpio.write(1,gpio.LOW);analog_gas = 0
tmr.alarm(2, 100, 1, function()
  analog_gas = adc.read(0)
  if analog_gas>300 then beep(3); gpio.write(1,gpio.HIGH)
  elseif analog_gas>30 then beep(2); gpio.write(1,gpio.HIGH)
  else gpio.write(1,gpio.LOW) end
end)
gpio.mode(2,gpio.OUTPUT);gpio.write(2,gpio.LOW);gas_digital = 0
function tengoGas(b)
  if b==0 then beep(4);gas_digital = 1;gpio.write(2,gpio.HIGH)
    tmr.alarm(4,1500,0,function()
      gas_digital=0;gpio.write(2,gpio.LOW)
    end)
  end
end
gpio.mode(3,gpio.OUTPUT);gpio.write(3,gpio.LOW);flama_digital = 0
function tengoFlama(b)
  if b==0 then
    beep(5);flama_digital = 1;gpio.write(3,gpio.HIGH)
    tmr.alarm(5,1500,0,function()
      flama_digital=0;gpio.write(3,gpio.LOW)
    end)
  end
end
```

Listado 4.6: Código fuente del dispositivo detector de gases

```
gpio.mode(5,gpio.INPUT)
tmr.alarm(3,100,1,function() tengoGas(gpio.read(5)) end)
gpio.mode(12,gpio.INPUT)
tmr.alarm(6,100,1,function() tengoFlama(gpio.read(12)) end)
srv=net.createServer(net.TCP)
srv:listen(80,function(conn)
  conn:on("receive",function(conn,comando)
    print(comando)
    if comando=="\r\n" or comando=="\n" then
      print("line")
    --GET VALUES
    elseif comando=="get values" then
      pct = dht_errors.."/"..dht_cycles
      xxx = "#temp|"..tmp.."|humd|"..hmd.."|gas_ana|"..analog_gas
      yyy = "|gas_dig|"..gas_digital.."|flama|"..flama_digital.."#\n"
      conn:send(xxx..yyy)
      pct = nil
      xxx = nil
      yyy = nil
    --HTTP GET /
    elseif stringStarts(comando, "GET / ") then
      conn:send("HTTP/1.1 200 OK\n\nIntelidomo")
    --HTTP GET /OTROS
    elseif stringStarts(comando, "GET /") then
      conn:send("HTTP/1.1 404 NOT FOUND\n\nIntelidomo")
    --CUALQUIER OTRA COSA
    else
      conn:send("#error|invalid command#\n")
    end
    conn:close()
    comando = nil
  end)
end)
```

Listado 4.7: Continuación de código fuente del dispositivo detector de gases

```
wifi.setmode(wifi.STATION)
wifi.sta.config("SSID","PASSWORD")
--Declara pines como salidas
gpio.mode(1,OUTPUT);gpio.mode(2,OUTPUT);
gpio.mode(3,OUTPUT);gpio.mode(4,OUTPUT);
--vars
max_steps=1500; current_steps=0;
--Ir hacia adelante
function forward(steps,conn)
  local var = 0; xpct = -1
  tmr.alarm(5,10,1,function()
    setStep(1,0,0,1);delay(1000)
    setStep(0,0,0,1);delay(1000)
    setStep(0,0,1,1);delay(1000)
    setStep(0,0,1,0);delay(1000)
    setStep(0,1,1,0);delay(1000)
    setStep(0,1,0,0);delay(1000)
    setStep(1,1,0,0);delay(1000)
    setStep(1,0,0,0)
    current_steps = current_steps+1
    var = var + 1
    pct = current_steps*100/max_steps
    if var==steps then
      tmr.stop(5);setStep(0,0,0,0);send(conn,pct);close(conn)
    end
  end)
end
--Establecer paso
function setStep(w1, w2, w3, w4)
  write(1,w1);write(2,w2);write(3,w3);write(4,w4)
end
--Ir a
function goto(goto_pct,conn)
  if goto_pct<0 then goto_pct=0 end
  if goto_pct>100 then goto_pct=100 end
  curr_pct = current_steps*100/max_steps
  goto_steps = max_steps*goto_pct/100
  steps = goto_steps-current_steps
  if steps>0 then forward(steps,conn) end
  if steps<0 then backwards(steps*-1,conn) end
end
--Bajar cortina
gpio.mode(5,gpio.INPUT,gpio.PULLUP)
gpio.trig(5,"down",function(l)
  curr_pct = current_steps*100/max_steps;goto(curr_pct-25)
end)
```

Listado 4.8: Fragmento del código fuente del dispositivo de control de cortinas

```
srv=net.createServer(net.TCP)
srv:listen(80,function(conn)
  conn:on("receive",function(conn,comando)
    if comando=="\r\n" or comando=="\n" then
      print("line")
    --GET POSITION
    elseif comando=="get position" then
      pct = current_steps*100/max_steps
      conn:send("#position|" .. pct .. "#\n")
      conn:close()
    --SET POSITION
    elseif stringStarts(comando,"set position ") then
      parts = split(comando," ")
      if table.maxn(parts) ~= 3 then
        conn:send("#error|invalid set#\n")
      else
        local x = tonumber(parts[3])
        goto(x,conn)
      end
    --SET MAXSTEPS
    elseif stringStarts(comando,"set maxsteps ") then
      parts = split(comando," ")
      if table.maxn(parts) ~= 3 then
        conn:send("#error|invalid set#\n")
      else
        local new_max = tonumber(parts[3])
        max_steps = new_max
        conn:send("#maxsteps|" .. max_steps .. "#\n")
      end
      conn:close()
    --SAVE CONFIGURATION
    elseif comando=="save config" then
      respaldar()
      conn:send("#info|done#\n")
      conn:close()
    --HTTP GET
    elseif stringStarts(comando, "GET / ") then
      conn:send("HTTP/1.1 404 NOT FOUND\n\nIntelidomo")
      conn:close()
    --CUALQUIER OTRA COSA
    else
      conn:send("#error|invalid command#\n")
      conn:close()
    end
    comando = nil
  end)
end)
```

Listado 4.9: Continuación de código fuente del dispositivo de control de cortinas

```
wifi.setmode(wifi.STATION)
wifi.sta.config("SSID","PSW")
gpio.mode(1,gpio.OUTPUT);gpio.write(1,gpio.LOW);
gpio.mode(2,gpio.OUTPUT);gpio.write(2,gpio.LOW);
pwm.setup(1,1000,1023);pwm.start(1);pwm.setduty(1,0);
pwm.setup(2,1000,1023);pwm.start(2);pwm.setduty(2,0);
current_pct1 = 0;current_pct2 = 0
function goto(which,pct)
  duty = 1023*pct/100
  pwm.setduty(which,duty)
  if which==1 then current_pct1 = pct
  else current_pct2 = pct end
end
srv=net.createServer(net.TCP)
srv:listen(80,function(conn)
  conn:on("receive",function(conn,comando)
    --GET
    if stringStarts(comando,"get ") then
      parts = split(comando," ")
      length = table.maxn(parts)
      luz = parts[2];
      if luz=="light1" then
        conn:send("#light1|"..current_pct1.."#\n")
      else
        conn:send("#light2|"..current_pct2.."#\n")
      end
    --SET
    elseif stringStarts(comando,"set ") then
      parts = split(comando," ")
      length = table.maxn(parts)
      luz = parts[2];
      pct_str = parts[4];
      pct_number = tonumber(pct_str)
      if luz=="light1" then
        goto(1,pct_number)
        conn:send("#light1|"..current_pct1.."#\n")
      else
        goto(2,pct_number)
        conn:send("#light2|"..current_pct2.."#\n")
      end
    --HTTP GET
    elseif stringStarts(comando, "GET /") then
      conn:send("HTTP/1.1 404 NOT FOUND\n\nIntelidomo")
    end
    conn:close()
  end)
end)
```

Listado 4.10: Código fuente del dispositivo de luces regulables

Capítulo 5

Conclusiones y recomendaciones

El objetivo de este capítulo es mostrar las conclusiones obtenidas con el desarrollo de esta tesis, algunas recomendaciones y una lista de mejoras consideradas como parte de trabajo a futuro.

5.1. Conclusiones

La ingeniería de Líneas de Productos de Software ayuda a acelerar el desarrollo de una familia completa de productos. El proceso de desarrollo obliga a tener una plataforma de software bien diseñada, totalmente escalable, confiable y robusta. Por supuesto, esto requiere mayor tiempo de análisis e implementación, sin embargo, una vez concluida la plataforma, añadir las demás características se facilita y se acelera el ritmo de desarrollo, logrando reducir tiempos de implementación y por lo tanto costos. Mejorar la plataforma de software conlleva a mejorar toda la familia de productos, por lo que la calidad de los productos siempre va mejorando.

DeltaJ fue un gran facilitador para implementar la línea de productos, su sintaxis y forma de trabajar es muy flexible. Se detectaron tres pequeñas desventajas que quizá lleguen a disuadir a los desarrolladores de utilizarlo: (1) la sintaxis es compatible con la versión 1.5 de Java, lo que quita ciertos atajos o funcionalidades del código; (2) al editar un delta, la opción de autocompletar código no está disponible, lo que a veces resulta incómodo; y (3) los errores de semántica se detectan hasta generar los productos. Las ventajas superan con creces estas desventajas: manipular

el código fuente se hace de manera sencilla e intuitiva, agregando, modificando o quitando clases, campos, importaciones, métodos y constructores; definir las características y los productos se hace en un sólo archivo de configuración, indicando las características que pertenece a cuál producto donde cualquier combinación válida es un nuevo producto; es posible generar toda una familia de productos con un clic y el proceso de generación tarda cuestión de segundos.

Actualmente está en auge el concepto del *Internet de las Cosas* con muchos dispositivos disponibles para automatizar ciertas áreas del hogar, sin embargo nunca cooperan entre sí y sus interfaces están *desperdigadas* entre aplicaciones y artefactos. Los productos de domótica desarrollados integran todos los dispositivos en una interfaz común y los abstrae por lo que son (luces, termómetros, televisiones), sin que a las aplicaciones cliente les importe cómo se logra controlar el dispositivo o todo lo que se tenga que hacer para lograr. La integración también permite que los dispositivos colaboren entre sí, realizando acciones en un dispositivo según el estado de otro o varios otros dispositivos.

El configurador de Líneas de Productos de Software desarrollado para automatizar la documentación es un *bonus* que obtuvo buenos resultados, con él, se logró crear una documentación acertada para cada uno de los productos generados. Cuenta con la misma flexibilidad de DeltaJ, es posible crear documentación para cualquier combinación válida de características. Cubre las áreas de características, definición de productos, requerimientos, diseño y pruebas, todo con una interfaz amigable y completando el ciclo del proceso de desarrollo indicado por el marco de trabajo. El hecho de ser capaz de abrir y guardar en diferentes archivos distintos proyectos, le da la capacidad de utilizarse como herramienta CASE y apoyar en el desarrollo de distintas Líneas de Productos de Software.

Los dispositivos desarrollados para cubrir el área domótica han demostrado ser confiables y trabajar sin interrupciones durante días.

5.2. Recomendaciones

Aún queda mucho trabajo por hacer en la LPS de domótica, hay gran variedad de dispositivos en el mercado a los cuales es posible agregar soporte.

Hay muchas mejoras por hacer en el configurador desarrollado, es posible agregar soporte para más de un estándar de definición de características (OVM o FODA), o agregar funcionalidad para que también se soporte el uso de diagramas de casos de uso y sea capaz de hacer con ellos las mismas operaciones de combinación que se hacen con los diagramas de clases. No todas las formas disponibles en los diagramas de clases se soportan aún por el configurador, falta implementar la representación gráfica de figuras de paquetes, navegación en asociaciones o nombres de las asociaciones, entre otras cosas. Quizá un enfoque orientado a aspectos permita tener mejor control sobre la ejecución de las pruebas, ya que a veces la bitácora de las pruebas se capturaban cuando la siguiente comenzaba, provocando malos reportes en la exportación.

Integrar el configurador desarrollado con el configurador de DeltaJ es un buen reto recomendado, se considera factible y así se solucionarán desventajas de integración, ya se están utilizando dos herramientas distintas donde la definición de características y productos está en ambas partes y no hay cooperación entre sí. Integrar los dos configuradores en una sola solución facilitará aún más el desarrollo de Líneas de Productos de Software completas.

Si se desea estar al tanto de nuevas funcionalidades agregadas o noticias futuras sobre este proyecto, se recomienda estar al tanto de *https://intelidomo.com* ya que ahí se planea publicar tutoriales y guías que permitan a casi cualquier persona con conocimientos básicos de programación, integrar servicios de domótica en sus hogares utilizando algunos de los recursos desarrollados en este proyecto de tesis.

Productos académicos

En congreso internacional

Victor Jassiel Díaz Velásquez, Ulises Juárez Martínez, Luis Ángel Reyes Hernández, Beatriz Alejandra Olivares Zepahua. *Líneas de Productos de Software y DeltaJ para mejorar el desarrollo de software en la región de Orizaba*, II Congreso Internacional de Robótica y Computación CIRC 2015, Instituto Tecnológico de La Paz, ISBN: 978-607-95534-8-7, pp. 182-187, México, 2015.

Anexos

A. Encuesta de definición de productos

El primer objetivo para la definición de productos implementando el esquema de Kano es realizar la identificación de los requerimientos del cliente y para ello se realizó la siguiente encuesta aplicada a 64 Hoteles clientes de la empresa Microcalli del Golfo.

- ¿Cómo consideraría el hecho de poder controlar diversos aparatos electrónicos a través de un solo sistema inteligente?
- ¿Cuáles son los procesos o aparatos electrónicos que le gustaría automatizar y controlar de manera remota?
- ¿Cree usted que esta automatización le ayudaría de alguna manera?
- ¿Qué inconvenientes o dificultades cree Usted que podría haber al implementar este tipo de automatización al hotel?
- ¿Cuáles serían las consideraciones que Usted tomaría al adquirir un producto de este tipo?
- ¿Qué características le gustaría que tuvieran esta automatización?
- ¿Estaría dispuesto a implementar estas automatizaciones en su hotel?
- ¿Cuáles son los beneficios que cree Usted adquirir junto con el producto?
- ¿Considera alguna otra forma de automatización conveniente para su hotel?

- ¿Tiene algún comentario o sugerencia a cerca de estas nuevas automatizaciones?

El segundo objetivo propuesto dentro del esquema de Kano es la realización de dos cuestionarios, uno de preguntas funcionales y el otro de preguntas no funcionales para cada uno de los requerimientos, para ello se aplicaron las siguientes encuestas de los productos de la LPS.

Preguntas funcionales:

- Suponga que es posible abrir y cerrar las persianas de una habitación automáticamente, ¿Qué piensa de ello?
- Imagine que puede ahorrar energía eléctrica dentro del hotel incluso cuanto tenga ocupación llena. ¿Qué opina de ello?
- ¿Qué pensaría si desde el sistema de recepción pudieran configurarse los despertadores en las habitaciones?
- ¿Cuál sería su opinión si las habitaciones se climatizaran automáticamente cuando estén ocupadas?
- Si pudiera ahorrar energía automatizando las luces de una habitación. ¿Qué opinaría al respecto?
- ¿Cómo consideraría poder dar un mensaje de bienvenida al huésped cuando llegue a la habitación y poder ofrecerle los servicios con los que cuenta el Hotel?
- Piense que el control de calderas se automatiza según las zonas del hotel con habitaciones ocupadas ¿Qué piensa de ello?
- ¿Cuál sería su reacción si pudiera controlar diversos productos automáticos como los ya mencionados desde un dispositivo móvil (celular, tablet o laptop)?
- Si pudiera tener alertas de seguridad, incendios, fugas de gas, entre otros, y poder tener una alerta vía SMS, Sonora o una llamada automática a la policía o bomberos ¿Qué tan importante seria para Usted este tipo de servicios?

- ¿Cómo se sentiría si pudiera recibir las solicitudes de servicios a huéspedes de manera centralizada a través del sistema llevando un control de cada solicitud y tiempo de respuesta?

Preguntas no funcionales:

- Suponga que no se puede abrir y cerrar las persianas de una habitación automáticamente, ¿Qué piensa de ello?
- ¿Qué pensaría si no pudiera ahorrar energía dentro del hotel?
- ¿Qué pensaría si desde el sistema de recepción no pudieran configurarse los despertadores en las habitaciones?
- ¿Cuál sería su opinión si las habitaciones no pudieran climatizarse automáticamente cuando estén ocupadas?
- Si no pudiera ahorrar energía automatizando las luces de una habitación ¿Qué opinaría al respecto?
- Si no pudiera dar un mensaje de bienvenida al huésped cuando llegue a la habitación y poder ofrecerle los servicios con los que cuenta el Hotel. ¿Qué opinaría?
- Piense que el control de calderas no es automático según las zonas del hotel con habitaciones ocupadas ¿Qué piensa de ello?
- ¿Cuál sería su reacción si no pudiera controlar los diversos productos automáticos como los ya mencionados desde un dispositivo móvil (Celular, Tablet o Laptop) y todo tuviera que ser de forma manual?
- Si no pudiera tener alertas de seguridad, incendios, fugas de gas, entre otros, y poder tener una alerta vía SMS, Sonora o una llamada automática a la policía o bomberos ¿Qué opinaría de esto?
- ¿Qué pensaría si no pudiera recibir las solicitudes de servicios a huéspedes de manera centralizada a través del sistema llevando un control de cada solicitud y tiempo de respuesta?

Se enviaron las encuestas a 120 hoteles contemplados dentro del padrón de hoteles, de los cuales se presenta menor a un 50 % el porcentaje de contestación. A pesar de esto, todos los hoteles que participaron en la encuesta tuvieron una aceptación de la LPS en su totalidad, dando a esto una certeza sobre las necesidades del cliente por las características presentadas ante los beneficios que recibirá después de su implementación.

Los productos que se están manejando dentro de la LPS fue resultado de haber realizado un estudio de mercado en donde se podía mostrar los productos de los distintos distribuidores dentro del sector hotelero a lo largo del país, así se demuestra la preferencia de los clientes por el ahorro de energía entre los productos mostrados.

Una vez visto el tipo de producto de mayor preferencia para los hoteles dentro de la LPS, se observa cuáles son los beneficios que esperan recibir ante la implementación de los ya mencionados. Estas características tienen que ser primordiales para la comercialización de los productos tanto para los hoteles como para residencias u oficinas, la cual es una de los principales objetivos al poder realizar componentes adaptables a cualquier tipo de aplicación.

Las inconveniencias que los hoteles consideran al implementar este tipo de productos es principalmente el costo de inversión, debido a que se debe de considerar la compatibilidad que se tenga esta automatización con el producto existente en los hoteles como por ejemplo equipos de aire acondicionado, televisiones, iluminación, etc. Por lo que al no ser compatibles se tendría que invertir en nuevos equipos que puedan adaptarse a este esquema de automatización.

B. Documentación del dominio

La ingeniería de dominio es la primera etapa definida en el marco de trabajo empleado para la realización de Líneas de Productos de Software. En esta etapa se define lo común y lo variable de una línea de productos de software así como las aplicaciones de software planeadas y el alcance de ellas. Se definen y construyen artefactos reutilizables que cumplan con la variabilidad deseada. Consta de cinco subprocesos en cada uno de los cuales se detalla y refina la variabilidad definida en el subproceso anterior, también se retroalimenta la factibilidad de realizar la variabilidad planeada. El principal objetivo de esta etapa es el desarrollo de la plataforma de software.

Fase de requerimientos

A continuación se describen los requerimientos presentes en la **plataforma**.

Objetivos

- G1.1: Ofrecer un sistema de control y automatización de diferentes dispositivos en un hogar.
- G1.2: Integrar dichos dispositivos en una red y de manera inteligente hacer cooperar sus funciones.
- G1.3: Controlar el comportamiento de los dispositivos y realizar tareas complejas con poca o nula intervención humana.
- G1.4: Ofrecer una interfaz vistosa e intuitiva para el usuario final.
- G1.5: Aumentar la comodidad del usuario en su hogar.
- G1.6: Mejorar la seguridad al ofrecer herramientas que permitan tomar acciones rápidas a escenarios de peligro.
- G1.7: Prevenir peligros para la salud humana.
- G1.8: Reducir los costos al evitar el uso innecesario de recursos que no se estén utilizando.

Rasgos deseados

- F1: Control de los dispositivos del hogar.

Requerimientos

- R1: El sistema debe tener una arquitectura fácilmente escalable y flexible que permita agregar diferentes abstracciones de dispositivos domóticos como convenga.

A continuación se describen los requerimientos aplicados al cumplirse la condición **cortina**.

Objetivos

- G8: Facilitar el control de las cortinas del hogar.

Rasgos deseados

- F8: Control de cortinas.

Requerimientos

- R8.2: La cortina se marcará como "No disponible" cuando el servidor falle al contactar el dispositivo de cortina o el dispositivo retorne respuestas no válidas.
- R8.1: El servidor incluirá clases abstractas que permitan representar una cortina, de la cual extenderán las clases que implementen un dispositivo de control de cortina concreto.
- R8.3: Las aplicaciones cliente mostrarán (por cada cortina) un ícono que permita identificar al dispositivo, una etiqueta que indique el número de cortina y un botón deslizable para cambiar la posición de la cortina.

A continuación se describen los requerimientos aplicados al cumplirse la condición **tv**.

Objetivos

- G6: Facilitar el control de televisiones.

Rasgos deseados

- F6: Control de TV's.

Requerimientos

- R6.1: El servidor incluirá clases abstractas que representen una televisión del hogar, de la cuál extenderán las clases que implementen el control concreto de una televisión específica.
- R6.2: El servidor incluirá clases abstractas que representen las operaciones básicas a ejecutar sobre una televisión, como encenderla y apagarla, cambiar de canal o presionar una tecla de un control remoto.
- R6.4: Las aplicaciones cliente mostrarán (por cada televisión) un ícono que permita identificar el tipo de dispositivo, una etiqueta que indique su tipo y número y un botón que permita mostrar u ocultar los controles adicionales (las teclas del control, por ejemplo).

- R6.3: La televisión se marcará como “No disponible” cuando el servidor falle al contactar el dispositivo que controle la televisión o el dispositivo retorne respuestas no válidas.

A continuación se describen los requerimientos aplicados al cumplirse la condición **domopi & temperatura**.

Rasgos deseados

- F33: Obtención de la temperatura de un área a través de una Raspberry Pi.

Requerimientos

- R33.1: La Raspberry Pi debe ser capaz de poder leer el estado de uno o más sensores de temperatura.
- R33.2: Las respuestas para los estados de sensores de temperatura de la Raspberry Pi se basa en los siguientes ejemplos #TEMP|33.1|TEMPn|27.5#

A continuación se describen los requerimientos aplicados al cumplirse la condición **domopi & presencia**.

Rasgos deseados

- F32: Conocer la presencia (o ausencia) de personas en un área a través de una Raspberry Pi.

Requerimientos

- R32.1: La Raspberry Pi debe ser capaz de poder leer el estado de uno o más sensores de presencia.
- R32.2: Las respuestas para los estados de sensores de presencia de la Raspberry Pi se basa en los siguientes ejemplos #PIR|SI|PIRn|NO#

A continuación se describen los requerimientos aplicados al cumplirse la condición **fuego**.

Objetivos

- G9: Contar con una alarma temprana que permita conocer si se genera fuego en un área específica para atender el siniestro antes de que ocurran daños mayores.

Rasgos deseados

- F9: Generar alarmas de fuego.

Requerimientos

- R9.1: El servidor incluirá clases que especialicen las clases de la característica alarma y permita diferenciar una alarma de fuego.
- R9.2: Las aplicaciones cliente contarán con un ícono personalizado para la alarma de tipo fuego.

A continuación se describen los requerimientos aplicados al cumplirse la condición **nodepi & luces**.

Objetivos

- G41: Control de una o dos luces de bajo voltaje a través de un microcontrolador NodeMCU y un controlador de motores DC.

Requerimientos

- R41.1: Un dispositivo MCU se programará especialmente para controlar el encendido y apagado de luces, así como su intensidad. El servidor domótico se comunicará con él a través del protocolo TCP/IP.
- R41.2: El dispositivo aceptará comandos del tipo SET [light1|light2] to [valor], donde valor es un número entero de 0 a 100 indicando la luminosidad de la luz requerida.
- R41.3: El dispositivo aceptará comandos del tipo GET [light1|light2] y devolverá una cadena con el formato #[light1|light2]|[valor]#, donde valor indica la potencia de luz actual.

A continuación se describen los requerimientos aplicados al cumplirse la condición **presencia**.

Objetivos

- G3: Conocer si hay presencia de algún individuo en un área determinada.

Rasgos deseados

- F3: Lectura de presencia.

Requerimientos

- R3.1: El servidor incluirá clases abstractas que representen la ausencia o presencia de una persona en un área, de la cual extenderán las clases que implementen un dispositivo sensor de presencia concreto.
- R3.2: Las aplicaciones cliente mostrarán (por cada dispositivo de presencia) un ícono que permita identificar el tipo de dispositivo, una etiqueta que indique su tipo y número y otra etiqueta más que muestre el valor actual de la lectura.
- R3.3: El dispositivo sensor de presencia se marcará como "No disponible" cuando el servidor falle al contactar el dispositivo o el dispositivo retorne respuestas no válidas.

A continuación se describen los requerimientos aplicados al cumplirse la condición **luz**.

Objetivos

- G2: Facilitar el encendido y/o apagado de luces.

Rasgos deseados

- F2: Control de luces.

Requerimientos

- R2.1: El servidor incluirá clases abstractas que representen una luz, de la cual extenderán las clases que implementen un dispositivo de luz concreto.
- R2.3: Las aplicaciones cliente mostrarán (por cada luz) un ícono que permita identificar que se trata de una luz, una etiqueta que indique el número de luz y un botón conmutable para encender o apagar la luz.

- R2.2: La luz se marcará como “No disponible” cuando el servidor falle al contactar el dispositivo de luz o el dispositivo retorne respuestas no válidas.

A continuación se describen los requerimientos aplicados al cumplirse la condición **domopi**.

Objetivos

- G30: Ofrecer un conjunto de soluciones que satisfagan ciertas características de un ambiente domótico, esto a través de una Raspberry Pi, que gracias a su poder de procesamiento es capaz de encargarse de distintos dispositivos, distintas áreas y soló con un dispositivo.

Rasgos deseados

- F30: Control de diversos dispositivos a través de una Raspberry Pi programada por uno mismo.

Requerimientos

- R30.1: Programar un servidor TCP/IP en la(s) Raspberry Pi sobre el puerto 8585 (recomendado) para lograr la comunicación con el dispositivo.
- R30.2: Tener la capacidad de instalar más de un servidor TCP/IP sobre puertos diferentes para simular control de áreas distintas con una misma Raspberry Pi.
- R30.7: La Raspberry Pi debe mantener la conexión abierta y avisar de cambios de estado en los dispositivos en tiempo real en el mismo formato especificado.
- R30.3: El servidor TCP/IP de la Raspberry Pi debe aceptar comandos del tipo SET KEY = VALUE para realizar cambios de estado en los dispositivos.
- R30.4: El servidor TCP/IP de la Raspberry Pi debe aceptar comandos del tipo GET KEY para obtener los valores de un dispositivo o sensor específico.
- R30.5: El servidor TCP/IP de la Raspberry Pi debe devolver el estado de todos los dispositivos que maneja al enviársele el comando GET ALL.

- R30.6: El formato de las respuestas que la Raspberry debe enviar al pedírseles datos, debe tener el siguiente formato: #KEY1|VALUE1|KEY2|VALUE2|KEYn|VALUEn#
- R30.8: Después de que el servidor de domótica envíe un comando o el servidor de la Raspberry dé una respuesta, se debe incluir un salto de línea al final.
- R30.9: La Raspberry Pi debe se capaz de controlar más de un dispositivo del mismo tipo, enumerándolos (opcionalmente si sólo es uno) del 1 al 9.

A continuación se describen los requerimientos aplicados al cumplirse la condición **alarma**.

Objetivos

- G7.1: Ofrecer una manera eficaz de crear alarmas y conocer sus eventos.
- G7.2: Conocer a tiempo eventos que amenacen la seguridad del usuario o del hogar para tomar acciones preventivas o correctivas.

Rasgos deseados

- F7: Activación de alarmas.

Requerimientos

- R7.1: El servidor incluirá clases abstractas que representen una alarma cualquiera, de la cual extenderán las clases que implementen un dispositivo de alarma concreto.
- R7.3: Al generarse un evento en una alarma, este debe guardarse en una bitácora para posterior consulta o análisis.
- R7.4: Al generarse un evento de alarma, el servidor debe avisar a la mayor brevedad posible a los clientes conectados.
- R7.6: Los clientes conectados al servidor, al recibir un evento de alarma, deben ejecutar algún mecanismo (un sonido, un mensaje SMS, por ejemplo) que permita al usuario conocer lo más pronto posible la situación y este sea capaz de realizar alguna acción preventiva o correctiva si se requiere.

- R7.5: Las aplicaciones cliente mostrarán (por cada alarma) un ícono que permita identificar el tipo de dispositivo, una etiqueta que indique su tipo y número y otra etiqueta más que muestre el valor actual que la alarma represente.

- R7.2: La alarma se marcará como "No disponible" cuando el servidor falle al contactar el dispositivo de alarma o el dispositivo retorne respuestas no válidas.

A continuación se describen los requerimientos aplicados al cumplirse la condición **gas**.

Objetivos

- G10: Contar con una alarma temprana que permita conocer en un área hay concentraciones peligrosas de gases como monóxido de carbono, dióxido de carbono, gas LP, butano, metano, etcétera; y atender la causa.

Rasgos deseados

- F10: Generar alarmas de gases peligrosos.

Requerimientos

- R10.1: El servidor incluirá clases que especialicen las clases de la característica alarma y permita diferenciar una alarma de gases peligrosos.

- R10.2: Las aplicaciones cliente contarán con un ícono personalizado para la alarma de gases peligrosos.

A continuación se describen los requerimientos aplicados al cumplirse la condición **arduino-rodas & temperatura**.

Objetivos

- G23: Conocer la temperatura de un área específica a través de un Arduino.

Rasgos deseados

- F23: Lectura de temperatura a través de un Arduino.

Requerimientos

- R23: El servidor domótico agregará la implementación de un sensor de temperatura y actualizará su estado al enviar el comando GET /TEMP al dispositivo Arduino de Rodas y obtener respuesta.

A continuación se describen los requerimientos aplicados al cumplirse la condición **domopi & tv**.

Rasgos deseados

- F35: Control de una o más televisiones a través de una Raspberry Pi.

Requerimientos

- R35.1: La Raspberry Pi debe tener la librería LIRC habilitada y configurada para el envío de comandos a televisiones a través de diodos infrarrojos.
- R35.2: La Raspberry Pi debe tener un servidor especial para el control de televisiones basado en el estándar REST.
- R35.3: Las peticiones HTTP para simular la presión de una tecla de un control remoto de una televisión, se enviarán a la Raspberry haciendo peticiones a la URL http://ip-raspberry/KEY/tecla_a_enviar
- R35.4: Las peticiones HTTP para solicitar un canal se enviarán a la Raspberry haciendo peticiones a la URL http://ip-raspberry/CHANNEL/número_canal

A continuación se describen los requerimientos aplicados al cumplirse la condición **nodepi & temperatura**.

Requerimientos

- R44: El servidor domótico creará una implementación de un sensor de temperatura y actualizará su estado cada vez que la cadena retornada por el dispositivo NodeMCU contenga la clave temp.

A continuación se describen los requerimientos aplicados al cumplirse la condición **nodepi & gas**.

Requerimientos

- R46: El servidor domótico creará una implementación de una alarma de gases peligrosos y actualizará su estado cada vez que la cadena retornada por el dispositivo NodeMCU contenga la clave *gas_ana* o *gas_dig*.

A continuación se describen los requerimientos aplicados al cumplirse la condición **nodepi & cortina**.

Requerimientos

- R47.1: Un dispositivo NodeMCU será programado especialmente para el control de cortinas a través de un motor de paso por paso, esto sobre el protocolo TCP/IP
- R47.2: El dispositivo aceptará los comandos set position [valor] donde valor será un número entero del 0 al 100 para indicar la posición requerida de la cortina.
- R37.3: El dispositivo aceptará los comandos get position y devolverá una cadena con el formato #position|[valor]# donde valor será un número entero del 0 al 100 que indica la posición actual de la cortina.
- R47.3: El servidor domótico creará una implementación de una cortina y actualizará su estado cada cinco minutos para conocer la posición actual de la cortina en caso de que haya sido controlada manualmente.
- R47.4: El dispositivo aceptará comando adicionales que permitan configurarlo adecuadamente a cualquier tamaño de ventana, como cambiar la posición inicial y posición maxima.

A continuación se describen los requerimientos aplicados al cumplirse la condición **arduino-rodas**.

Objetivos

- G20: Ofrecer un dispositivo de control de ciertas características a través de un microcontrolador Arduino programado por la empresa Rodas Computación S.A. de C.V.

Rasgos deseados

- F20: Control de diversos dispositivos a través de un controlador Arduino programado por la empresa Rodas Computación S.A. de C.V.

Requerimientos

- R20: El servidor domótico agregará las clases necesarias para conectarse a un dispositivo Arduino programado por la empresa Rodas Computación S.A. de C.V. y utilizar sus recursos.

A continuación se describen los requerimientos aplicados al cumplirse la condición **temperatura**.

Objetivos

- G4: Facilitar el conocimiento de la temperatura de un área.

Rasgos deseados

- F4: Lectura de temperatura.

Requerimientos

- R4.3: El sensor de temperatura se debe marcar como “No disponible” cuando el servidor no logre contactar al dispositivo u obtenga una respuesta no válida.
- R4.2: Las aplicaciones cliente mostrarán (por cada dispositivo sensor de temperatura) un ícono que permita identificar el tipo de dispositivo, una etiqueta que indique su tipo y número y otra etiqueta más que muestre el valor actual de la temperatura.
- R4.1: El servidor incluirá clases abstractas que representen la ponderación de la temperatura de un área, de la cual extenderán las clases que implementen un dispositivo sensor de temperatura concreto.

A continuación se describen los requerimientos aplicados al cumplirse la condición **humedad**.

Objetivos

- G5: Facilitar el conocimiento de la humedad relativa aproximada de un área.

Rasgos deseados

- F5: Lectura de humedad relativa.

Requerimientos

- R5.1: El servidor incluirá clases abstractas que representen la ponderación de la humedad relativa de un área, de la cual extenderán las clases que implementen un dispositivo sensor de humedad concreto.
- R5.2: Las aplicaciones cliente mostrarán (por cada dispositivo sensor de humedad) un ícono que permita identificar el tipo de dispositivo, una etiqueta que indique su tipo y número y otra etiqueta más que muestre el valor actual de la humedad relativa.
- R5.3: El sensor de humedad se debe marcar como "No disponible" cuando el servidor no logre contactar al dispositivo u obtenga una respuesta no válida.

A continuación se describen los requerimientos aplicados al cumplirse la condición **nodepi & humedad**.

Requerimientos

- R43: El servidor domótico creará una implementación de un sensor de humedad y actualizará su estado cada vez que la cadena retornada por el dispositivo NodeMCU contenga la clave humd.

A continuación se describen los requerimientos aplicados al cumplirse la condición **domopi & luz**.

Rasgos deseados

- F31: Control de luces a través de una Raspberry Pi.

Requerimientos

- R31.1: La Raspberry Pi debe ser capaz de controlar una o más luces, sin importar el mecanismo (transistores, *mosfets* o relevadores, etcétera).
- R31.2: Las respuestas para los estados de luz debe basarse en los siguientes ejemplos: #LUZ1|ON|LUZ2OFF|LUZn|VALUE#
- R31.3: El comando para cambiar el estado de una luz puede aceptar los valores ON o 1 para encender y OFF o 0 para apagar. Ejemplo: SET LUZ1 = ON ó SET LUZ1 = 1

A continuación se describen los requerimientos aplicados al cumplirse la condición **arduino-rodas & luces**.

Requerimientos

- R21.1: El servidor domótico agregará la implementación de una luz por cada dispositivo Arduino y actualizará cierto tiempo enviando el comando GET /LIGHT.
- R21.2: El servidor domótico enviará los comandos GET /LIGHTON y GET /LIGHTOFF para encender y apagar la luz respectivamente al dispositivo Arduino de Rodas.

A continuación se describen los requerimientos aplicados al cumplirse la condición **arduino-rodas & luz**.

Objetivos

- G21: Controlar una luz a través de relevadores con el microcontrolador Arduino.

Rasgos deseados

- F21: Control de una luz a través de un Arduino.

A continuación se describen los requerimientos aplicados al cumplirse la condición **nodepi & temperatura & humedad & gas & flama**.

Requerimientos

- R42.1: Un dispositivo NodeMCU es programado especialmente para leer los valores de sensores humedad, temperatura, ponderación de gases peligrosos y detección de flama.
- R42.2: El dispositivo aceptará el comando get values y devolverá el resultado del procesamiento de los valores de los sensores anteriormente descritos en una cadena con el formato #KEY1|VALUE1|KEYn|VALUEn# a través del protocolo TCP/IP.
- R42.3: El servidor domótico agregará las clases necesarias para conectarse con el dispositivo NodeMCU descrito y recuperar la cadena con los valores del dispositivo.

A continuación se describen los requerimientos aplicados al cumplirse la condición **domopi & humedad**.

Rasgos deseados

- F34: Obtención de la humedad relativa de un área a través de una Raspberry Pi.

Requerimientos

- R34.1: La Raspberry Pi debe ser capaz de poder leer el estado de uno o más sensores de humedad.
- R34.2: Las respuestas para los estados de sensores de humedad de la Raspberry Pi se basa en los siguientes ejemplos #HUMD|58.6|HUMDn|75.6#

A continuación se describen los requerimientos aplicados al cumplirse la condición **arduino-rodas & presencia**.

Objetivos

- G22: Conocer la presencia (o ausencia) de personas en un área a través de un Arduino.

Rasgos deseados

- F22: Lectura de presencia a través de un Arduino.

Requerimientos

- R22: El servidor domótico agregará la implementación de un sensor de presencia y actualizará su estado al enviar el comando GET /PIR al dispositivo Arduino de Rodas y obtener respuesta.

A continuación se describen los requerimientos aplicados al cumplirse la condición **nodepi**.

Objetivos

- G40: Ofrecer un conjunto de soluciones que satisfagan ciertas características de un ambiente domótico, esto a través de un microcontrolador NodeMCU Lua ESP8266, este dispositivo incluye funciones parecidas a un Arduino e incorpora un chip ESP12 que permite su conexión con redes WiFi, ideal para el internet de las cosas.

Rasgos deseados

- F40: Control de diversos dispositivos a través de un microcontrolador NodeMCU ESP8266 programado por uno mismo.

Requerimientos

- R40.1: Los dispositivos NodeMCU tendrán un servidor TCP/IP que permita al servidor domótico interactuar con él y realizar operaciones o leer valores que se requieran.
- R40.2: El servidor domótico incluirá clientes específicos para cada tipo de controlador Nodepi (dispositivo NodeMCU) y agregar las clases necesarias para hacer la abstracción.

A continuación se describen los requerimientos aplicados al cumplirse la condición **nodepi & flama**.

Requerimientos

- R45: El servidor domótico creará una implementación de una alarma de fuego y actualizará su estado cada vez que la cadena retornada por el dispositivo NodeMCU contenga la clave flama.

A continuación se describen los requerimientos aplicados al cumplirse la condición **domopi & luces**.

Objetivos

- G31: Control de diversas luces a través de relevadores con una Raspberry Pi.

C. Documentación del producto *Completo*

La ingeniería de aplicación es la segunda etapa definida en el marco de trabajo empleado para la implementación de Líneas de Productos de Software. En esta etapa se explotan las posibilidades de la plataforma y la variabilidad definidas en la ingeniería de dominio, se reutilizan los insumos de dominio tanto como sea posible, se documentan los artefactos de la aplicación (como requerimientos, arquitectura, componentes y pruebas) y se relacionan con los artefactos elaborados en la ingeniería de dominio, se enlaza la variabilidad con los requerimientos de una aplicación para extender la arquitectura, componentes y casos de uso, por último se estima el impacto de las diferencias de los requerimientos de dominio y aplicación en las arquitecturas, componentes y pruebas. El principal objetivo de esta etapa es el desarrollo de un producto específico planeado en la línea de productos de software.

Fase de requerimientos

A continuación se describe la definición textual de los requerimientos para el producto.

Objetivos

- G1.1: Ofrecer un sistema de control y automatización de diferentes dispositivos en un hogar.
- G1.2: Integrar dichos dispositivos en una red y de manera inteligente hacer cooperar sus funciones.
- G1.3: Controlar el comportamiento de los dispositivos y realizar tareas complejas con poca o nula intervención humana.
- G1.4: Ofrecer una interfaz vistosa e intuitiva para el usuario final.

- G1.5: Aumentar la comodidad del usuario en su hogar.
- G1.6: Mejorar la seguridad al ofrecer herramientas que permitan tomar acciones rápidas a escenarios de peligro.
- G1.7: Prevenir peligros para la salud humana.
- G1.8: Reducir los costos al evitar el uso innecesario de recursos que no se estén utilizando.
- G2: Facilitar el encendido y/o apagado de luces.
- G3: Conocer si hay presencia de algún individuo en un área determinada.
- G4: Facilitar el conocimiento de la temperatura de un área.
- G5: Facilitar el conocimiento de la humedad relativa aproximada de un área.
- G6: Facilitar el control de televisiones.
- G7.1: Ofrecer una manera eficaz de crear alarmas y conocer sus eventos.
- G7.2: Conocer a tiempo eventos que amenacen la seguridad del usuario o del hogar para tomar acciones preventivas o correctivas.
- G8: Facilitar el control de las cortinas del hogar.
- G9: Contar con una alarma temprana que permita conocer si se genera fuego en un área específica para atender el siniestro antes de que ocurran daños mayores.
- G10: Contar con una alarma temprana que permita conocer en un área hay concentraciones peligrosas de gases como monóxido de carbono, dióxido de carbono, gas LP, butano, metano, etcétera; y atender la causa.
- G20: Ofrecer un dispositivo de control de ciertas características a través de un microcontrolador Arduino programado por la empresa Rodas Computación S.A. de C.V.
- G21: Controlar una luz a través de relevadores con el microcontrolador Arduino.

- G22: Conocer la presencia (o ausencia) de personas en un área a través de un Arduino.
- G23: Conocer la temperatura de un área específica a través de un Arduino.
- G30: Ofrecer un conjunto de soluciones que satisfagan ciertas características de un ambiente domótico, esto a través de una Raspberry Pi, que gracias a su poder de procesamiento es capaz de encargarse de distintos dispositivos, distintas áreas y soló con un dispositivo.
- G40: Ofrecer un conjunto de soluciones que satisfagan ciertas características de un ambiente domótico, esto a través de un microcontrolador NodeMCU Lua ESP8266, este dispositivo incluye funciones parecidas a un Arduino e incorpora un chip ESP12 que permite su conexión con redes WiFi, ideal para el internet de las cosas.

Rasgos deseados

- F1: Control de los dispositivos del hogar.
- F2: Control de luces.
- F3: Lectura de presencia.
- F4: Lectura de temperatura.
- F5: Lectura de humedad relativa.
- F6: Control de TV's.
- F7: Activación de alarmas.
- F8: Control de cortinas.
- F9: Generar alarmas de fuego.
- F10: Generar alarmas de gases peligrosos.
- F20: Control de diversos dispositivos a través de un controlador Arduino programado por la empresa Rodas Computación S.A. de C.V.

- F21: Control de una luz a través de un Arduino.
- F22: Lectura de presencia a través de un Arduino.
- F23: Lectura de temperatura a través de un Arduino.
- F30: Control de diversos dispositivos a través de una Raspberry Pi programada por uno mismo.
- F31: Control de luces a través de una Raspberry Pi.
- F32: Conocer la presencia (o ausencia) de personas en un área a través de una Raspberry Pi.
- F33: Obtención de la temperatura de un área a través de una Raspberry Pi.
- F34: Obtención de la humedad relativa de un área a través de una Raspberry Pi.
- F35: Control de una o más televisiones a través de una Raspberry Pi.
- F40: Control de diversos dispositivos a través de un microcontrolador NodeMCU ESP8266 programado por uno mismo.

Requerimientos

- R1: El sistema debe tener una arquitectura fácilmente escalable y flexible que permita agregar diferentes abstracciones de dispositivos domóticos como convenga.
- R2.1: El servidor incluirá clases abstractas que representen una luz, de la cual extenderán las clases que implementen un dispositivo de luz concreto.
- R2.2: La luz se marcará como “No disponible” cuando el servidor falle al contactar el dispositivo de luz o el dispositivo retorne respuestas no válidas.
- R2.3: Las aplicaciones cliente mostrarán (por cada luz) un ícono que permita identificar que se trata de una luz, una etiqueta que indique el número de luz y un botón conmutable para encender o apagar la luz.

- R3.1: El servidor incluirá clases abstractas que representen la ausencia o presencia de una persona en un área, de la cual extenderán las clases que implementen un dispositivo sensor de presencia concreto.
- R3.2: Las aplicaciones cliente mostrarán (por cada dispositivo de presencia) un ícono que permita identificar el tipo de dispositivo, una etiqueta que indique su tipo y número y otra etiqueta más que muestre el valor actual de la lectura.
- R3.3: El dispositivo sensor de presencia se marcará como "No disponible" cuando el servidor falle al contactar el dispositivo o el dispositivo retorne respuestas no válidas.
- R4.1: El servidor incluirá clases abstractas que representen la ponderación de la temperatura de un área, de la cual extenderán las clases que implementen un dispositivo sensor de temperatura concreto.
- R4.2: Las aplicaciones cliente mostrarán (por cada dispositivo sensor de temperatura) un ícono que permita identificar el tipo de dispositivo, una etiqueta que indique su tipo y número y otra etiqueta más que muestre el valor actual de la temperatura.
- R4.3: El sensor de temperatura se debe marcar como "No disponible" cuando el servidor no logre contactar al dispositivo u obtenga una respuesta no válida.
- R5.1: El servidor incluirá clases abstractas que representen la ponderación de la humedad relativa de un área, de la cual extenderán las clases que implementen un dispositivo sensor de humedad concreto.
- R5.2: Las aplicaciones cliente mostrarán (por cada dispositivo sensor de humedad) un ícono que permita identificar el tipo de dispositivo, una etiqueta que indique su tipo y número y otra etiqueta más que muestre el valor actual de la humedad relativa.
- R5.3: El sensor de humedad se debe marcar como "No disponible" cuando el servidor no logre contactar al dispositivo u obtenga una respuesta no válida.

- R6.1: El servidor incluirá clases abstractas que representen una televisión del hogar, de la cuál extenderán las clases que implementen el control concreto de una televisión específica.
- R6.2: El servidor incluirá clases abstractas que representen las operaciones básicas a ejecutar sobre una televisión, como encenderla y apagarla, cambiar de canal o presionar una tecla de un control remoto.
- R6.3: La televisión se marcará como "No disponible" cuando el servidor falle al contactar el dispositivo que controle la televisión o el dispositivo retorne respuestas no válidas.
- R6.4: Las aplicaciones cliente mostrarán (por cada televisión) un ícono que permita identificar el tipo de dispositivo, una etiqueta que indique su tipo y número y un botón que permita mostrar u ocultar los controles adicionales (las teclas del control, por ejemplo).
- R7.1: El servidor incluirá clases abstractas que representen una alarma cualquiera, de la cual extenderán las clases que implementen un dispositivo de alarma concreto.
- R7.2: La alarma se marcará como "No disponible" cuando el servidor falle al contactar el dispositivo de alarma o el dispositivo retorne respuestas no válidas.
- R7.3: Al generarse un evento en una alarma, este debe guardarse en una bitácora para posterior consulta o análisis.
- R7.4: Al generarse un evento de alarma, el servidor debe avisar a la mayor brevedad posible a los clientes conectados.
- R7.5: Las aplicaciones cliente mostrarán (por cada alarma) un ícono que permita identificar el tipo de dispositivo, una etiqueta que indique su tipo y número y otra etiqueta más que muestre el valor actual que la alarma represente.
- R7.6: Los clientes conectados al servidor, al recibir un evento de alarma, deben ejecutar algún mecanismo (un sonido, un mensaje SMS, por ejemplo) que permita al usuario conocer lo más pronto posible la situación y este sea capaz de realizar alguna acción preventiva o correctiva si se requiere.

- R8.1: El servidor incluirá clases abstractas que permitan representar una cortina, de la cual extenderán las clases que implementen un dispositivo de control de cortina concreto.
- R8.2: La cortina se marcará como “No disponible” cuando el servidor falle al contactar el dispositivo de cortina o el dispositivo retorne respuestas no válidas.
- R8.3: Las aplicaciones cliente mostrarán (por cada cortina) un ícono que permita identificar al dispositivo, una etiqueta que indique el número de cortina y un botón deslizable para cambiar la posición de la cortina.
- R9.1: El servidor incluirá clases que especialicen las clases de la característica alarma y permita diferenciar una alarma de fuego.
- R9.2: Las aplicaciones cliente contarán con un ícono personalizado para la alarma de tipo fuego.
- R10.1: El servidor incluirá clases que especialicen las clases de la característica alarma y permita diferenciar una alarma de gases peligrosos.
- R10.2: Las aplicaciones cliente contarán con un ícono personalizado para la alarma de gases peligrosos.
- R20: El servidor domótico agregará las clases necesarias para conectarse a un dispositivo Arduino programado por la empresa Rodas Computación S.A. de C.V. y utilizar sus recursos.
- R22: El servidor domótico agregará la implementación de un sensor de presencia y actualizará su estado al enviar el comando GET /PIR al dispositivo Arduino de Rodas y obtener respuesta.
- R23: El servidor domótico agregará la implementación de un sensor de temperatura y actualizará su estado al enviar el comando GET /TEMP al dispositivo Arduino de Rodas y obtener respuesta.

- R30.1: Programar un servidor TCP/IP en la(s) Raspberry Pi sobre el puerto 8585 (recomendado) para lograr la comunicación con el dispositivo.
- R30.2: Tener la capacidad de instalar más de un servidor TCP/IP sobre puertos diferentes para simular control de áreas distintas con una misma Raspberry Pi.
- R30.3: El servidor TCP/IP de la Raspberry Pi debe aceptar comandos del tipo SET KEY = VALUE para realizar cambios de estado en los dispositivos.
- R30.4: El servidor TCP/IP de la Raspberry Pi debe aceptar comandos del tipo GET KEY para obtener los valores de un dispositivo o sensor específico.
- R30.5: El servidor TCP/IP de la Raspberry Pi debe devolver el estado de todos los dispositivos que maneja al enviársele el comando GET ALL.
- R30.6: El formato de las respuestas que la Raspberry debe enviar al pedírseles datos, debe tener el siguiente formato: #KEY1|VALUE1|KEY2|VALUE2|KEYn|VALUEn#
- R30.7: La Raspberry Pi debe mantener la conexión abierta y avisar de cambios de estado en los dispositivos en tiempo real en el mismo formato especificado.
- R30.8: Después de que el servidor de domótica envíe un comando o el servidor de la Raspberry dé una respuesta, se debe incluir un salto de línea al final.
- R30.9: La Raspberry Pi debe se capaz de controlar más de un dispositivo del mismo tipo, enumerándolos (opcionalmente si sólo es uno) del 1 al 9.
- R31.1: La Raspberry Pi debe ser capaz de controlar una o más luces, sin importar el mecanismo (transistores, *mosfets* o relevadores, etcétera).
- R31.2: Las respuestas para los estados de luz debe basarse en los siguientes ejemplos: #LUZ1|ON|LUZ2OFF|LUZn|VALUE#
- R31.3: El comando para cambiar el estado de una luz acepta los valores ON o 1 para encender y OFF o 0 para apagar. Ejemplo: SET LUZ1 = ON ó SET LUZ1 = 1

- R32.1: La Raspberry Pi debe ser capaz de poder leer el estado de uno o más sensores de presencia.
- R32.2: Las respuestas para los estados de sensores de presencia de la Raspberry Pi se basa en los siguientes ejemplos #PIR|SI|PIRn|NO#
- R33.1: La Raspberry Pi debe ser capaz de poder leer el estado de uno o más sensores de temperatura.
- R33.2: Las respuestas para los estados de sensores de temperatura de la Raspberry Pi se basa en los siguientes ejemplos #TEMP|33.1|TEMPn|27.5#
- R34.1: La Raspberry Pi debe ser capaz de poder leer el estado de uno o más sensores de humedad.
- R34.2: Las respuestas para los estados de sensores de humedad de la Raspberry Pi se basa en los siguientes ejemplos #HUMD|58.6|HUMDn|75.6#
- R35.1: La Raspberry Pi debe tener la librería LIRC habilitada y configurada para el envío de comandos a televisiones a través de diodos infrarrojos.
- R35.2: La Raspberry Pi debe tener un servidor especial para el control de televisiones basado en el estándar REST.
- R35.3: Las peticiones HTTP para simular la presión de una tecla de un control remoto de una televisión, se enviarán a la Raspberry haciendo peticiones a la URL http://ip-raspberry/KEY/tecla_a_enviar
- R35.4: Las peticiones HTTP para solicitar un canal se enviarán a la Raspberry haciendo peticiones a la URL http://ip-raspberry/CHANNEL/número_canal
- R37.3: El dispositivo aceptará los comandos get position y devolverá una cadena con el formato #position|[valor]# donde valor será un número entero del 0 al 100 que indica la posición actual de la cortina.

- R40.1: Los dispositivos NodeMCU tendrán un servidor TCP/IP que permita al servidor domótico interactuar con él y realizar operaciones o leer valores que se requieran.
- R40.2: El servidor domótico incluirá clientes específicos para cada tipo de controlador Nodepi (dispositivo NodeMCU) y agregar las clases necesarias para hacer la abstracción.
- R43: El servidor domótico creará una implementación de un sensor de humedad y actualizará su estado cada vez que la cadena retornada por el dispositivo NodeMCU contenga la clave humd.
- R44: El servidor domótico creará una implementación de un sensor de temperatura y actualizará su estado cada vez que la cadena retornada por el dispositivo NodeMCU contenga la clave temp.
- R46: El servidor domótico creará una implementación de una alarma de gases peligrosos y actualizará su estado cada vez que la cadena retornada por el dispositivo NodeMCU contenga la clave *gas_ana* o *gas_dig*.
- R47.1: Un dispositivo NodeMCU será programado especialmente para el control de cortinas a través de un motor de paso por paso, esto sobre el protocolo TCP/IP
- R47.2: El dispositivo aceptará los comandos set position [valor] donde valor será un número entero del 0 al 100 para indicar la posición requerida de la cortina.
- R47.3: El servidor domótico creará una implementación de una cortina y actualizará su estado cada cinco minutos para conocer la posición actual de la cortina en caso de que haya sido controlada manualmente.
- R47.4: El dispositivo aceptará comando adicionales que permitan configurarlo adecuadamente a cualquier tamaño de ventana, como cambiar la posición inicial y posición maxima.

Bibliografía

[1] M. Aiello, "The role ofweb services at home," in *Proceedings of the Advanced Int'L Conference on Telecommunications and Int'L Conference on Internet and Web Applications and Services*, ser. AICT-ICIW '06, 2006, pp. 164–170.

[2] K. Pohl, G. Böckle, and F. J. v. d. Linden, *Software Product Line Engineering: Foundations, Principles and Techniques*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2005.

[3] I. Schaefer, L. Bettini, F. Damiani, and N. Tanzarella, "Delta-oriented programming of software product lines," in *Proceedings of the 14th International Conference on Software Product Lines: Going Beyond*, ser. SPLC'10, 2010, pp. 77–91.

[4] J. Koscielny, S. Holthusen, I. Schaefer, S. Schulze, L. Bettini, and F. Damiani, "Deltaj 1.5: Delta-oriented programming for java 1.5," in *Proceedings of the 2014 International Conference on Principles and Practices of Programming on the Java Platform: Virtual Machines, Languages, and Tools*, ser. PPPJ '14, 2014, pp. 63–74.

[5] S. Thaker, D. Batory, D. Kitchin, and W. Cook, "Safe composition of product lines," in *Proceedings of the 6th International Conference on Generative Programming and Component Engineering*, ser. GPCE '07, 2007, pp. 95–104.

[6] C. Kästner, S. Apel, and M. Kuhlemann, "Granularity in software product lines," in *Proceedings of the 30th International Conference on Software Engineering*, ser. ICSE '08, 2008, pp. 311–320.

[7] D. Faust and C. Verhoef, "Software product line migration and deployment," *Software Practice and Experience, John Wiley & Sons, Ltd*, vol. 33, pp. 933–955, 2003.

[8] M. Alférez, J. a. Santos, A. Moreira, A. Garcia, U. Kulesza, J. a. Araújo, and V. Amaral, "Multi-view composition language for software product line requirements," in *Proceedings of the Second International Conference on Software Language Engineering*, ser. SLE'09, 2010, pp. 103–122.

[9] D. Bruns, V. Klebanov, and I. Schaefer, "Verification of software product lines with delta-oriented slicing," in *Proceedings of the 2010 International Conference on Formal Verification of Object-oriented Software*, ser. FoVeOOS'10, 2011, pp. 61–75.

[10] S. Schulze, O. Richers, and I. Schaefer, "Refactoring delta-oriented software product lines," in *Proceedings of the 12th Annual International Conference on Aspect-oriented Software Development*, ser. AOSD '13, 2013, pp. 73–84.

[11] C. Kästner, K. Ostermann, and S. Erdweg, "A variability-aware module system," *SIGPLAN Not.*, vol. 47, no. 10, pp. 773–792, Oct. 2012.

[12] W. Fenske, T. Thüm, and G. Saake, "A taxonomy of software product line reengineering," in *Proceedings of the Eighth International Workshop on Variability Modelling of Software-Intensive Systems*, ser. VaMoS '14, 2013, pp. 4:1–4:8.

[13] P. Anthonysamy and S. S. Somé, "Aspect-oriented use case modeling for software product lines," in *Proceedings of the 2008 AOSD Workshop on Early Aspects*, ser. EA '08, 2008, pp. 5:1–5:8.

[14] M. A. Babar, L. Chen, and F. Shull, "Managing variability in software product lines," *IEEE Softw.*, vol. 27, no. 3, pp. 89–91, 94, May 2010.

[15] M. Matinlassi, "Comparison of software product line architecture design methods: Copa, fast, form, kobra and qada," in *Proceedings of the 26th International Conference on Software Engineering*, ser. ICSE '04, 2004, pp. 127–136.

[16] K. Czarnecki, M. Antkiewicz, C. H. P. Kim, S. Lau, and K. Pietroszek, "Model-driven software product lines," in *Companion to the 20th Annual ACM SIGPLAN Conference on Object-oriented Programming, Systems, Languages, and Applications*, ser. OOPSLA '05, 2005, pp. 126–127.

[17] D. Muthig and T. Patzke, "Generic implementation of product line components," in *Revised Papers from the International Conference NetObjectDays on Objects, Components, Architectures, Services, and Applications for a Networked World*, ser. NODe '02, 2003, pp. 313–329.

[18] J. McGregor, "Agile software product lines, deconstructed," in *Journal of Object Technology*, LLC, USA, 2008.

[19] I. Reinhartz-Berger and K. Figl, "Comprehensibility of orthogonal variability modeling languages: The cases of cvl and ovm," in *Proceedings of the 18th International Software Product Line Conference - Volume 1*, ser. SPLC '14, 2014, pp. 42–51.

[20] V. Miori, D. Russo, and M. Aliberti, "Domotic technologies incompatibility becomes user transparent," *Commun. ACM*, vol. 53, no. 1, pp. 153–157, Jan. 2010.

[21] O. Haugen, B. Moller-Pedersen, J. Oldevik, G. Olsen, and A. Svendsen, "Adding standardized variability to domain specific languages," in *Software Product Line Conference, 2008. SPLC '08. 12th International*, 2008, pp. 139–148.

[22] (2015) Raspberry pi: Teach, learn and make. [Online]. Available: https://www.raspberrypi.org/

[23] (2015) Projects to make with raspberry pi. [Online]. Available: https://www.raspberrypi.org/resources/make/

[24] (2015) Nodemcu web page. [Online]. Available: http://nodemcu.com/index_en.html

[25] (2015) How to choose a gas sensor. [Online]. Available: http://www.seeedstudio.com/wiki/How_to_choose_A_Gas_Sensor

[26] (2015) Modulación por ancho de pulsos. [Online]. Available: https://es.wikipedia.org/wiki/Modulación_por_ancho_de_pulsos

[27] T. Karvinen, K. Kimmo, and V. Valtokari, *Make: Sensors*, B. Jepson, Ed. Sebastopol, Canada: Maker Media, Inc., 2014.