Proyecto Final de Carrera

Desarrollo de un Sistema de Gestión del Hogar Digital aplicando un enfoque Dirigido por Modelos

Julio del 2006, Valencia

Estefanía Serral Asensio
eserral@dsic.upv.es

Dirigido por: *Dr. Vicente Pelechano Ferragud*
pele@dsic.upv.es
Javier Muñoz Ferrara
jmunoz@dsic.upv.es

Índice

1. Introducción 5
2. Contexto tecnológico 9
  2.1. EIB 9
    2.1.1. Modo básico de funcionamiento 10
    2.1.2. EIB Tool Software (ETS) 12
      2.1.2.1. Funciones ETE 13
      2.1.2.2. Módulos del Software ETS2 13
      2.1.2.3. Utilización 14
      2.1.2.4. Diseño 14
  2.2. OSGI 14
    2.2.1. Especificación 15
3. Método dirigido por modelos aplicado en el caso de estudio 19
4. PervML: Modelado conceptual de sistemas pervasivos 21
  4.1. Vista del Analista 21
  4.2. Vista del Arquitecto 23
5. Framework de Soporte a PervML 25
  5.1. Estructura del Framework 25
    5.1.1. Capa Lógica 26
    5.1.2. Capa de interfaz 27
  5.2. Estrategia Global de Ejecución 29
    5.2.1. Secuencia de acciones 29
    5.2.2. Ejecución de operaciones 33
  5.3. Instanciación del Framework 36
    5.3.1. Creación de los Proveedores de Enlace 36
    5.3.2. Creación de los Componentes e Interacciones 37
6. Modelado del sistema pervasivo mediante PervML 40
  6.1. Requisitos del sistema 40
  6.2. Modelado del sistema 42
    6.2.1. El Modelo de Servicios 42
    6.2.2. El Modelo Estructural (Los Servicios del Sistema) 59
    6.2.3. El Modelo de Interacción 65
    6.2.4. El Modelo de Proveedores de Enlace 66
    6.2.5. Especificación Estructural de los componentes 72

6.2.6. Especificación Funcional 76
7. Drivers EIB 82
7.1. ¿Por qué EIB? 82
7.2. Instalación 82
7.3. Configuración mediante ETS2 86
7.4. Estrategia de Implementación 91
7.5. Ejemplos 93
7.5.1. GradualLamp 94
7.5.2. WallSwitch 98
7.5.3. Listener 101
8. Drivers simulados 104
8.1. Descripción del framework para drivers simulados 104
8.1.1. Estructura del framework 104
8.1.2. Infraestructura para el desarrollo de dispositivos virtuales 105
8.1.3. Uso del framework 106
8.1.3.1. Creación de nuevos tipos de dispositivos 106
8.1.3.2. Creación de una nueva instancia de un tipo de dispositivo 107
8.2. Ejemplos de drivers desarrollados 108
8.2.1. GradualLamp 109
8.2.2. WallSwitch 111
9. Drivers software: SMS 113
10. Interfaces de usuario 117
10.1. Interfaz para PDA 117
10.2. Interfaz Web 121
11. Conclusiones 122
11.1. Trabajos futuros 122
11.2. Publicaciones 122
12. Referencias 124

Índice de figuras

Figura 1. Distribución de los bits que forman la dirección física 11
Figura 2. Tipos de datos EIB 12
Figura 3. Arquitectura del sistema 19
Figura 4. Pasos para aplicar el método propuesto 20
Figura 5. Ejemplo de modelado de la vista del analista.................................... 23
Figura 6. Ejemplos de los modelos para la vista del arquitecto........................... 24
Figura 7. Estrategia propuesta por las Factorías Software y nuestra aplicación......... 25
Figura 8. Estructura del framework 26
Figura 9. Estructura de clases para soportar las capas de Comunicaciones y Servicios 27
Figura 10. Estructura de clases para soportar la capa de Interfaz 28
Figura 11. Pasos que se realizan ante un cambio en el entorno............................ 31
Figura 12. Pasos ante la solicitud de un servicio por parte del usuario 33
Figura 13. Estructura de clases para implementar los proveedores de enlace............ 36
Figura 14. Estructura de clases para implementar los componentes y las interacciones 37
Figura 15. Entorno real de implantación... 84
Figura 16. Distribución de los dispositivos en el primer piso................................ 85
Figura 17. Distribución de los dispositivos en el Segundo piso............................... 86
Figura 18. Diseño del proyecto en el ETS 87
Figura 19. Criterio para asignar direcciones de grupo 88
Figura 20. Módulo de puesta en marcha 91
Figura 21. Estructuración del framework 104
Figura 22. Clases de la infraestructura software del framework. 105
Figura 23. Configuración de las propiedades SMS..113
Figura 24. Freemarker 117
Figura 25. Interfaz para PDA del componente GardenLighting 121
Figura 26. Interfaz Web del componente GardenLighting...................................121

1. Introducción

El aumento del número y la complejidad de los sistemas pervasivos es un hecho. Este tipo de sistemas integran dispositivos y sistemas software para proporcionar cierta funcionalidad a los usuarios de un entorno físico determinado. Actualmente, dichos sistemas están siendo implantados cada vez más en el ámbito del hogar o, en general, de los edificios, proporcionando servicios para la automatización y control de los distintos elementos de una casa (iluminación, persianas, electrodomésticos, etc.) o proporcionando servicios de seguridad y detección de averías técnicas (detectores de presencia, cámaras de seguridad, detectores de humedad, sensores de gas, etc.).

En este proyecto se presenta el trabajo realizado para el desarrollo de un sistema software que proporciona servicios (control y monitorización de dispositivos, multimedia, comunicaciones, etc.) en el entorno de un hogar. Este trabajo se ha realizado dentro del grupo de investigación OO-Method, en el departamento de Sistema Informáticos y Computación. Este grupo se ocupa de definir métodos orientados a objetos para la construcción de software en ambientes de generación automática de código, permitiendo obtener sistemas software finales de mayor calidad y más rápidamente.

Actualmente no existe una gran difusión de los sistemas pervasivos para el hogar, debido en gran medida a que las tecnologías que se utilizaban hasta ahora todavía no se encontraban maduras y, por tanto, tenían un precio muy elevado. Esto está cambiando a gran velocidad debido a la aparición de nuevos factores. La creación de estándares específicos, el descenso de los precios, el interés de los constructores y de grandes compañías como Telefónica o Microsoft, hace prever un aumento en el número y complejidad de estos sistemas. Esto contrasta con el nivel de evolución de los métodos y técnicas para su desarrollo, ya que la mayoría utiliza tecnologías de muy bajo nivel de abstracción (programación de microcontroladores o de redes de control de dispositivos) que hacen que el desarrollo no sea eficiente y que el producto software no resulte escalable ni mantenible.

Para paliar estos problemas, en el contexto del grupo de investigación OO-Method se está desarrollando un método para el desarrollo de sistemas pervasivos que aplica las últimas tendencias en la ingeniería del software a este tipo de sistemas. En resumen, el método se basa en el lenguaje de modelado PervML. A partir de una descripción del sistema pervasivo realizada mediante este lenguaje, un compilador de modelos genera código Java que se instala en una pasarela residencial junto a un framework de implementación. Este código Java hace uso de una serie de drivers que deben ser desarrollados manualmente. Los drivers son una parte fundamental de la aplicación pervasiva, ya que son los encargados de interactuar con los dispositivos o sistemas software que finalmente proporcionarán la funcionalidad a los usuarios.

El Objetivo del presente Proyecto Final de Carrera es el desarrollo completo de un sistema pervasivo aplicando el método de desarrollo dirigido por modelos propuesto en el grupo de investigación OO-Method. Para ello ha sido necesario realizar el modelado del sistema utilizando el lenguaje PervML, el desarrollo de los drivers (tanto de acceso a los dispositivos como de acceso a sistemas software) y la correspondiente instalación física.

Así pues, en este proyecto se utilizará PervML para el desarrollo del sistema pervasivo. PervML es un lenguaje diseñado con el objetivo de proporcionar a los desarrolladores de sistemas pervasivos constructores adecuados para describir este tipo de sistemas de una manera precisa. PervML promueve la separación de roles, donde los desarrolladores pueden clasificarse en analistas y arquitectos. Los analistas describen los diferentes tipos de servicios proporcionados en el sistema así como la interacción entre ellos mediante tres modelos. En primer lugar, se ocupará el rol del analista y se definirán los modelos de Servicios, Estructural, y de Interacción. Básicamente, el sistema a implementar proporcionará servicios de cuatro tipos: (1) gestión de iluminación (de manera "inteligente" en algunos puntos de la casa), (2) gestión de multimedia (que permite reproducir archivos multimedia en varias ubicaciones), (3) gestión de la climatización (intentado no desperdiciar energía y aprovechando los recursos naturales) y (4) gestión de la seguridad (principalmente detección de intrusos y accidentes).

Los arquitectos completan la descripción del sistema especificando los dispositivos que implementan los servicios mediante tres modelos más. Ocupando el rol del arquitecto se definirá el Modelo de Proveedores de Enlace, la Especificación Estructural de los Componentes y la Especificación Funcional de los Componentes. Mediante estos modelos se asociarán a los servicios identificados los siguientes dispositivos: bombillas, interruptores, interruptores para iluminación gradual, detector de presencia, detector de movimiento, sensores de iluminación, sensor de temperatura, estación meteorológica, sensor de temperatura, actuador de persianas, etc.). Además, también serán asociados elementos software que formarán parte del sistema, como: mensajería instantánea, envió de correo, control de Media Player, etc. De este modo, la funcionalidad del sistema es descrita por los analistas de forma independiente a los dispositivos seleccionados por los arquitectos para implementarla.

Una vez los requisitos del sistema y su modelo conceptual estén construidos, se desarrollará un framework que facilite su implementación. El framework se implementará utilizando la tecnología OSGi, y encapsulará la estructura y funcionalidad compartida por los sistemas que produce el método. El desarrollo de este framework de implementación se ha realizado de manera conjunta con Carlos Cetina, ya que su Proyecto Final de Carrera también se ha realizado en un contexto similar.

Como se ha comentado anteriormente en esta introducción, una parte fundamental del sistema pervasivo son los drivers encargados de interactuar con los dispositivos físicos (bombillas, detectores, sensores, etc.) y elementos software (mensajería instantánea, envío de sms…) con el fin de implementar la funcionalidad proporcionada por los servicios. Estos drivers han de tratar cuestiones específicas de tecnología, por lo que deben ser desarrollados a mano. Una parte considerable de

este Proyecto Final de Carrera se ha invertido en el desarrollo de los drivers encargados de implementar la funcionalidad de los servicios del sistema. Es necesario tener en cuenta que el caso de estudio describe un entorno de un hogar con dos plantas y jardín en el que están implicados numerosos dispositivos, pero el entorno real de implantación ha sido un laboratorio de investigación con un espacio limitado y un número restringido (pero representativo) de dispositivos, por lo que ha sido necesario desarrollar dispositivos virtuales.

Por otra parte, Para interactuar con el sistema pervasivo, se ofrecen dos interfaces de usuario. Ambas son interfaces web: una para acceder desde navegadores de escritorio y otra para acceder desde navegadores de dispositivos móviles.

Resumiendo, las **aportaciones del presente proyecto** son las siguientes:

- El desarrollo de un sistema pervasivo de dimensiones muy cercanas a la realidad aplicando un método de producción de software dirigido por modelos.
- La aplicación del lenguaje PervML para el desarrollo de un caso de estudio real, lo cual contribuirá en la validación del lenguaje y permitirá detectar posibles carencias que necesiten ser solventadas.
- La implementación de un framework que aumenta el nivel de abstracción de la tecnología OSGi y proporciona constructores similares a los que define PervML.
- La implementación de drivers que permiten la gestión de los diferentes dispositivos vía software. Estos drivers son reutilizables para futuros proyectos.
- El desarrollo de diferentes interfaces para el control de los servicios proporcionados por el sistema pervasivo. Estas interfaces son reutilizables para futuros proyectos.

Al construir un sistema pervasivo utilizando un método dirigido por modelos en un ámbito de generación automática de código, se pretende demostrar las ventajas que presenta utilizar un método de estas características en cuanto a cambios imprevistos en los requisitos iniciales de la aplicación y en cuanto a la calidad del producto final producido.

Para describir el trabajo realizado en el presente Proyecto Final de Carrera, este documento se estructura en las siguientes secciones:

- **Sección 1**. Introducción
- **Sección 2**. Contexto Tecnológico: Se describen en esta sección las tecnologías que han sido necesarias para llevar a cabo el desarrollo del proyecto.
- **Sección 3.** Método dirigido por modelos aplicado en el caso de estudio: Se explican los pasos seguidos para llegar a obtener el caso de estudio en funcionamiento.

- **Sección 4.** PervML: Descripción del lenguaje de modelado utilizado para realizar la especificación del caso de estudio.

- **Sección 5.** Framework de soporte a PervML: Se explica en esta sección el desarrollo y la utilización del framework diseñado para servir de soporte a los sistemas especificados utilizando el lenguaje de modelado PervML.

- **Sección 6.** Modelado del sistema pervasivo mediante PervML: descripción del caso de estudio realizado mediante la especificación de los requisitos del sistema y el modelado del mismo utilizando los modelos propuestos por PervML.

- **Sección 7.** Drivers EIB: instalación, configuración e implementación de los drivers desarrollados para controlar los dispositivos EIB disponibles.

- **Sección 8.** Drivers simulados: se detalla en esta sección la estructura y el uso del framework desarrollado por otro proyecto final de carrera y utilizado para facilitar la simulación de sistemas pervasivos, así como los drivers simulados desarrollados específicamente para este caso de estudio.

- **Sección 9.** Drivers software: se detalla en esta sección la implementación de los drivers software desarrollados para el sistema pervasivo realizado.

- **Sección 10:** Interfaces de usuario: se explica en éste capítulo la realización de las dos interfaces de usuario desarrolladas, la interfaz Web y la interfaz para PDA.

- **Sección 11:** Conclusiones

- **Sección 12:** Referencias

2. Contexto tecnológico

En el ámbito de este proyecto se han necesitado utilizar diversas tecnologías. Se describen a continuación brevemente para facilitar la comprensión del trabajo realizado.

2.1. EIB

Para la instalación de los dispositivos se ha utilizado la tecnología EIB [1] (Bus de Instalación Europeo). El European Installation Bus o EIB es un sistema domótico desarrollado bajo los auspicios de la Unión Europea con el objetivo de crear un estándar europeo, con el suficiente número de fabricantes, instaladores y usuarios, que permita comunicarse a todos los dispositivos de una instalación eléctrica como: contadores, equipos de climatización, de seguridad, de gestión energética, electrodomésticos, etc. El EIB está basado en la estructura de niveles OSI y tiene una arquitectura descentralizada. Este estándar europeo define una relación extremo-a-extremo entre dispositivos que permite distribuir la inteligencia entre los sensores y los actuadores instalados en la vivienda.

Así pues, EIB constituye un completo sistema integrado de automatización y control de edificios y viviendas, destinado a la aplicación de soluciones gradualmente compatibles, flexibles y rentables. Debido a su versatilidad funcional, su uso no se reduce a las instalaciones simples y limitadas sino que también proporciona soluciones para el sector del edificio completo, ya que todos los componentes del bus tienen su propia inteligencia. Ante cambios de uso o reorganización del espacio, el EIB consigue una adaptación rápida y sin problemas, mediante cambio de parametrización (mediante el software ETS), de los componentes del bus, sin necesidad de un nuevo cableado, por ello, la instalación en un edificio se puede realizar de un modo más sencillo en un principio, y después se puede ampliar y modificar sin problemas.

El sistema EIB fue promovido desde 1990 por el grupo de fabricantes que engloba la EIBA (Asociación EIB) que tiene sede en Bruselas. Las tareas principales de esta asociación son:

- Fijar las directrices técnicas para el sistema y los productos EIB, así como establecer los procedimientos de ensayo y certificación de calidad.

- Distribuir el conocimiento y las experiencias de las empresas que trabajan sobre el EIB. Encarga a laboratorios de ensayo las pertinentes pruebas de calidad.

- Conceder a los productos EIB y a los fabricantes de estos una licencia de marca EIB con la que se podrán distribuir los productos.

- Colaborar activamente con otros organismos europeos o internacionales en todas las fases de la normalización y adaptación del sistema EIB a las normas vigentes.

- Liderar el proceso de convergencia de los tres buses europeos de más amplia difusión como son el propio EIB, el Batibus y el EHS (European Home System).

De esta manera, las empresas participantes en esta asociación garantizan que sus productos sean compatibles con el bus por lo que podemos emplear en la instalación EIB aparatos de distintos fabricantes con total interoperabilidad. Además, la EIBA provee de un set estándar de herramientas independientes del vendedor y componentes API para PC's con Windows. Según la EIBA (Asociación EIB) hay unos 10 millones de dispositivos EIB instalados por todo el mundo, unas 70.000 instalaciones, una gama de 4.500 productos diferentes, 113 empresas asociadas a la EIBA, y 70.000 instaladores cualificados.

2.1.1. Modo básico de funcionamiento

El EIB es un sistema preparado para utilizarse de forma descentralizada (todos sus componentes llevan un microprocesador), en el que se utiliza el protocolo de comunicación CSMA/CA (Carrier Sense Multiple Access with Collision Avoidance) para garantizar un acceso aleatorio y libre de colisiones al Bus. La transmisión se realiza en serie, de forma asíncrona y simétrica por los dos conductores, a una velocidad de 9.600 bit/seg.

La red del EIB se estructura de forma jerárquica y la unidad más pequeña se denomina línea, a la cual se pueden conectar hasta un máximo de 64 dispositivos. Las líneas se agrupan en áreas que se componen de una línea principal de la que pueden colgar hasta 15 líneas secundarias.

Las restricciones principales a las que debe someterse la instalación son:

- Ha de haber al menos una fuente de alimentación por línea
- La longitud total no puede superar los 1000 m.
- La distancia máxima entre la fuente de alimentación y un dispositivo sea menor de 350 m.
- La distancia máxima entre dispositivos no puede ser superior a los 750 m.
- La distancia entre dos fuentes de alimentación dentro de una misma línea debe ser inferior a los 200 m.

Se utilizan tres grupos de elementos funcionalmente diferenciados e interconectados: Línea Bus (cable bifilar, trenzado y apantallado que comunica a todos los componentes del Sistema, además de alimentarlos a 24 Vcc), sensores (convierten la pulsación de una tecla o una medición en una señal binaria digital –telegrama- que se envía a los correspondientes actuadores), actuadores (reciben las órdenes para actuar sobre los circuitos de potencia a ellos conectados).

Así, los sensores y actuadores se comunican entre sí mediante un par trenzado de baja tensión de seguridad. Este par proporciona la alimentación para la electrónica de los distintos componentes y también transmite la información entre ellos. Cada componente del sistema va dotado de un acoplador de bus, BCU. Cuando se acciona cualquiera sensor, se envía un telegrama.

Cada telegrama consta de los siguientes campos:

- Control (8 bits): indica inicio y prioridad de trama.

- Dirección del emisor (16 bits).
- Dirección del destinatario (16 bit +1 bit): el último bit indica si se trata de dirección física o de grupo.
- Contador (3 bits): se utiliza para funciones de enrutamiento, contando el número de saltos que ha dado el paquete.
- Longitud (4 bits): número de bits que ocupa LSDU.
- LSDU (Link Service Data Unit): que es la información a ser transmitida.(hasta 16x8 bits)
- Byte de comprobación.

Cada dispositivo tiene una dirección física de 16 bits asociada que le identifica unívocamente. La dirección de un dispositivo además define la localización de éste en la red. Cada dirección se divide en área, línea dentro del área, y número de dispositivo, como se muestra en la figura 1.

Figura 1. Distribución de los bits que forman la dirección física

Además de la dirección física, cada dispositivo puede tener una o más direcciones lógicas, denominadas direcciones de grupo. Todos los dispositivos que tengan la misma dirección de grupo reciben los mismos mensajes. Los sensores sólo pueden enviar telegramas a una dirección de grupo, mientras que los actuadores pueden tener varias direcciones de grupo, lo que les permite reaccionar a distintos sensores. Cualquier dispositivo de la red puede mandar telegramas a una dirección de grupo.

Dependiendo de la granularidad que el diseñador quiera dar a la red se puede seleccionar direcciones de grupo de nivel 2 o de nivel 3. Las direcciones de grupo de nivel 2 dividen la dirección en dos campos: grupo principal y subgrupo, mientras que las de nivel 3 separan la dirección en: grupo principal, grupo intermedio y subgrupo. Con el nivel 2 se obtienen 15 grupos principales con 2047 subgrupos cada grupo. Para el nivel 3 la división queda en 15 grupos principales, cada uno con 7 grupos intermedios de 255 subgrupos cada uno.

Para que dos dispositivos puedan comunicarse no solo deben conocer como localizarse entre sí sino también deben compartir una semántica común. Los datos intercambiados tienen que tener el mismo significado para los dos dispositivos. Para solucionarlo EIB ha definido el estándar EIS (EIB Interworking Standard).

En la siguiente figura se resumen los diferentes tipos de datos de que se disponen. Aunque el nombre es bastante indicativo de la semántica del tipo de dato, no significa que esté limitado exclusivamente a esa función. Por ejemplo el tipo de datos 2 (EIS type 2) que es regulación de la iluminación (dimming) también se puede

utilizar para control de la calefacción. Los datos se interpretarían como más caliente/frío en vez de más luminoso/oscuro.

EIS (EIB Interworking Standard)			
num. EIS	función EIB	nº de bytes	Descripción
EIS 1	interruptor (switching)	1 bit	encendido/apagado, habilitar/deshabilitar, alarma/no alarma, verdadero/falso
EIS 2	regulación (dimming)	4 bit	Se puede utilizar de 3 formas distintas: como interruptor, como valor relativo y como valor absoluto.
EIS 3	hora (time)	3 bytes	dia de la semana, hora, minutos y segundos.
EIS 4	fecha (date)	3 bytes	dia/mes/año (el margen es de 1990 a 2089).
EIS 5	valor (value)	2 bytes	Para enviar valores físicos con representación S,EEEE,MMMMMMMMMMM
EIS 6	escala (scaling)	8 bit	Se utiliza para transmitir valores relativos con una resolución de 8 bit. Ej. FF = 100 %
EIS 7	control motores (control drive)	1 bit	Tiene dos usos: Mover, arriba/abajo o extender/retraer y Paso a Paso.
EIS 8	prioridad (priority)	1 bit	Se utiliza en conjunción con EIS 1 ó EIS 7.
EIS 9	coma flotante (float value)	4 bytes	Codifica un número en coma flotante según el formato definido por el IEEE 754.
EIS 10	contador 16 bit (16b-counter)	2 bytes	Representa los valores de un contador de 16 bit (tanto con signo como sin signo).
EIS 11	contador 32 bit (32b-counter)	4 bytes	Representa los valores de un contador de 32 bit (tanto con signo como sin signo).
EIS 12	acceso (access)	4 bytes	Se usa para conceder accesos a distintas funciones.
EIS 13	Caracter ASCII (Character)	8 bit	Codifica según el formato ASCII
EIS 14	contador 8 bit (8b-counter)	8 bit	Representa los valores de un contador de 8 bits (tanto con signo como sin signo).
EIS 15	Cadena (Character String)	14 bytes	Transmite una cadena de caracteres ASCII de hasta 14 bytes.

Figura 2. Tipos de datos EIB

La programación de los dispositivos se realiza mediante el software ETS, explicado en la siguiente sección.

2.1.2. EIB Tool Software (ETS)

Tras la instalación, es necesario programar los sensores y actuadores, para ello se ha utilizado el programa ETS2 que provee la EIBA.

El ETS® es una herramienta de PC para el diseño, configuración y licencia de instalaciones EIB. El usuario crea y diseña un proyecto de instalación con el ETS, basado en plantillas de producto provistas por el fabricante distribuidas por el propio fabricante de componentes EIB. Una herramienta especial del kit del ETS permite al fabricante crear las plantillas y exportarlas como "Base de Datos de Producto" a fin de importarlas y utilizarlas como herramienta de diseño de proyectos con ETS.

El programa ETS2 ha sido diseñado, por consiguiente, con una estructura flexible, extensible y modular, para que también pueda dar soporte a los nuevos desarrollos de la tecnología EIB. Las funciones están claramente estructuradas y el programa contiene descripciones sobre la planificación y ejecución de proyectos concretos con ETS2. Además, el software es independiente del fabricante, con una disciplina de herramienta de software multi-PC para el sistema EIB.

2.1.2.1. Funciones ETE

La versión actual del software ETS es un sistema abierto. Consta de un entorno de desarrollo básico ETE (EIB Tool Environment), que proporciona las siguientes funciones:

- Interfaz de usuario
- Impresión
- Selección del idioma
- Acceso a bases de datos
- Acceso a instalaciones EIB por medio de RS232
- Importar/Exportar productos y proyectos
- Interfaz software para futuros módulos adicionales

Todos los módulos de software del programa ETS2 pueden acceder a estas funciones.

2.1.2.2. Módulos del Software ETS2

El software de aplicación ETS2 contiene los siguientes módulos:

- Configuración
- Diseño de Proyecto
- Puesta en Marcha/Test
- Administración de Proyectos
- Administración de Productos
- Informes

- Conversión

2.1.2.3. Utilización

El ETS® es utilizado con el fin de crear una base de datos de productos. Una vez se importan la información técnica de los productos que forman el sistema, se diseña la instalación sin estar conectado al bus insertando los productos de la librería en el proyecto afinando los parámetros específicos de cada producto.

El ETS® maneja productos certificados EIB de todos los proveedores de soluciones EIB. Corre bajo Windows 3.1x / 95-98, NT y XP. La versión actual es ETS®2 V1.3. A través de él, toda la información relevante del sistema está disponible en un modelo estándar de base de datos único y global.

2.1.2.4. Diseño

En la fase inicial, se debe tener en cuenta el tipo de edificio y su uso, las instalaciones del edificio y qué funciones van a tener, el tipo y frecuencia de los cambios de uso, requisitos especiales por parte del constructor, implicación en el coste del edificio, etc. Después hay que configurar los dispositivos, los pasos que deben seguirse son:

- asignación de las direcciones físicas (para la identificación unívoca de cada sensor o actuador en una instalación EIB);

- selección y programación (parametrización) del software de aplicación apropiado para los sensores y actuadores;

- asignación de direcciones de grupo (para unir las funciones de sensores y actuadores).

El programa realiza una comprobación automática del sistema y garantiza el cumplimiento de las reglas de instalación del Bus EIB. Identifica y localiza las posibles incoherencias que pueda haber. Además, establece automáticamente la lista de los productos, la lista de las direcciones e indica el contenido de las diferentes líneas, pudiendo ser impresas. Por último, el programa informático ETS exportar todos los datos necesarios para la puesta en servicio.

2.2. OSGI

El framework de implementación, descrito detalladamente en la Sección 5, ha sido implementado utilizando la tecnología para pasarelas residenciales OSGi [2].

La Open Service Gateway Initiative (OSGi) es una asociación de empresas creada con el objetivo de definir un estándar abierto para el desarrollo de pasarelas residenciales. Inicialmente fueron 15 las compañías que fundaron esta asociación, entre las que destacan: Sun Microsystems, IBM, Lucent Technologies, Motorola, Ericsson, Toshiba, Nortel Networks, Oracle, Philips, Sybase, Toshiba, entre otras. Ahora son más de 80 las empresas que pertenecen a esta asociación. Hay fabricantes de hardware o PCs, empresas de software, de sistemas de gestión corporativos, operadores de telecomunicaciones, hasta varias compañías eléctricas. Actualmente ya tiene socios españoles como Unión Fenosa y Telefónica I+D.

OSGi pretende ofrecer una arquitectura completa, de extremo-a-extremo, que cubra todas las necesidades del proveedor de servicios, del cliente y de cualquier dispositivo instalado en las viviendas. Define arquitectura software mínima necesaria para que todos los servicios se puedan ejecutar en la misma plataforma independientemente del hardware usado (microprocesador, memoria, periféricos, modems, etc.). De esta forma permite a cualquier fabricante decidir cómo y dónde instala este software en plataformas compatibles que sean capaces de proporcionar múltiples servicios en el ámbito de la vivienda.

Así pues, OSGI puede describirse como una colección de APIs basados en Java que permiten el desarrollo de servicios independientemente de la plataforma. Estas APIs permiten la compartición de los servicios, el manejo de datos, recursos y dispositivos, el acceso de clientes y la seguridad. Como la especificación OSGI está orientada a la capa de aplicación se puede complementar con cualquiera de los protocolos de comunicación y redes domóticas existentes, tales como Bluetooth, EIB, CEBus, Home API, HomePNA, HomePNP, HomeRF...

Describimos a continuación los aspectos más importantes de la especificación de OSGI.

2.2.1. Especificación

El núcleo de la especificación de OSGI es el framework que proporciona un entorno estandarizado para las aplicaciones (conocidas como bundles). Se divide en 4 capas:

- *Entorno de ejecución*: es la especificación del entorno de Java.
- *Módulos*: define la política de carga de clases
- *Gestión del ciclo de vida*: esta capa permite que los bundles puedan ser instalados, iniciados, parados, modificados o desinstalados dinámicamente. Un activador es la clase que se encargar de gestionar el ciclo de vida de un Bundle. Esta clase tiene unos métodos start y stop que son invocados cuando el Bundle se arranca y detiene, respectivamente.
- *Registro de Servicios*: permite que los bundles puedan compartir objetos. Un servicio no es más que un objeto Java, registrado como proveedor de una determinada interfaz, que puede ser simplemente un objeto software o puede representar un objeto del mundo real. Al ser registrado un servicio, se le pueden asociar una serie de propiedades que podrán ser utilizadas por otros servicios para realizar búsquedas concretas. Algunas de estas propiedades son obligatorias, como el PID, que sirve para identificar a un servicio durante toda su actividad en OSGi. Además permite que se puedan definir acciones a realizar cuando se registra o se deja de ofrecer el servicio.

La seguridad en OSGI está basada en el modelo de seguridad de Java y Java 2. El lenguaje limita ciertas construcciones, impidiendo por ejemplo errores del tipo overflow. La visibilidad del código queda restringida a los programadores. OSGI extiende el modelo para permitir clases privadas. El modelo de seguridad de Java 2 proporciona un modelo detallado para comprobar los accesos del código a los recursos. Además OSGI añade gestión dinámica completa de los permisos.

Por encima del framework se han desarrollado diversos servicios que se han especificado mediante una interfaz java y se han registrado como servicios utilizando el Registro de Servicios. Los clientes del servicio pueden encontrarlo en el registro, o reaccionar a él cuando aparece/desaparece. A continuación describimos brevemente los servicios que ofrece. Se debe resaltar que todos ellos han sido definidos de forma abstracta e implementada independientemente de los diferentes vendedores.

Los servicios que proporciona el framework son:

- *Permission admin*: los permisos de los bundles actuales o futuros pueden ser manipulados gracias a este servicio.

- *Package admin*: Los bundles comparten paquetes con clases y recursos. La modificación de los bundles puede que requiera recalcular las dependencias. Este servicio proporciona información sobre el estado actual de compartición de paquetes del sistema y puede refrescar las dependencias.

- *Start Leve*l: Conjunto de bundles que permiten fijar el valor del start level actual, permite asignar a un bundle un start level y consultar las opciones de configuración actuales.

- Los servicios que proporciona el sistema son:

- *Log Services*: el registro de información, warnings, información de debug o errores son manejados por este servicio. Éste recibe dichos registros y los reenvía convenientemente a los bundles suscritos a esta información.

- *Configuration Admin Service*: Éste servicio proporciona un modelo dinámico y flexible para configurar y obtener la información de configuración.

- *Device Access Service*: Mecanismo de OSGI que permite el descubrimiento y anuncio dinámico de dispositivos y de los servicios ofrecidos por éstos.

- *User Admin Service*: Éste servicio usa una base de datos con información del usuario para llevar a cabo la autentificación y la autorización.

- *IO Connector Service*: Implementa el paquete CDC/CLDC javax.microedition.io como un servicio. Permite a los bundles proporcionar esquemas de protocolo nuevos y alternativos.

- *Preferences Service*: servicio que proporciona acceso a una base de datos jerárquica de propiedades.

OSGI también ha definido servicios de protocolo que mapean un protocolo externo como un servicio de OSGI:

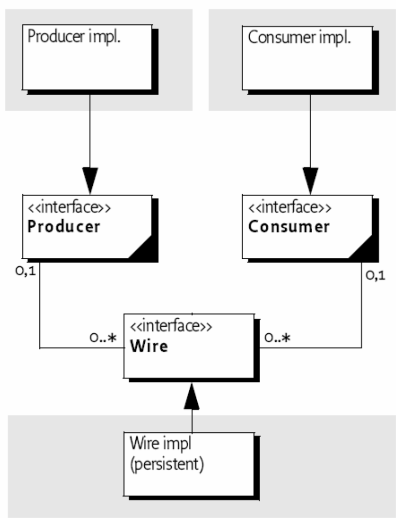
- *Http Service*: Es un servlet runner. Los bundles pueden proporcionar servlets que estarán disponibles mediante el protocolo http. La actualización dinámica que facilita la plataforma OSGI convierte a éste servicio en un servidor web que puede ser actualizado con nuevos servlets, incluso de forma remota, sin tener que ser reiniciado.

- *UPnP Service*: UPnP (Universal Plug and Play) es un estándar que emerge para la electrónica de consumidor. El servicio de OSGi UPnP mapea los dispositivos en una red de UPnP con un registro del servicio. Alternativamente, puede mapear los servicios de OSGi en la red de UPnP.

- *Jini Service*: Jini es un protocolo de red usado para encontrar servicios Jini en una red y descargar dichos servicios en un host como código java, pudiendo ejecutarlos.

Osgi define además otros dos servicios:

- *Wire admin Service*: este servicio permite conectar diferentes servicios tal y como esté definido en un fichero de configuración. Siguiendo una estrategia productor/consumidor permite establecer un canal de comunicación (wire) entre ambos, posibilitando el intercambio de objetos a través del mismo. La comunicación puede ser iniciada bajo demanda de cualquiera de los dos miembros, pero la información simple fluirá de manera unidireccional desde el productor al consumidor. El tipo de datos de la información que se transmite debe ser conocido tanto por el consumidor como por el productor desde un inicio. Para ello, al registrarse los participantes de un wire indicarán en la propiedad flavours qué tipo de datos van a manejar. En la siguiente figura se muestra el esquema de funcionamiento del mecanismo de Wires de OSGi.



- *XML Parser Service*: Permite a un bundle localizar un parser con las propiedades deseadas y compatible con JAXP.

OSGi es aplicable en muy diversos ámbitos, ya que permite la cooperación eficiente de múltiples componentes en una sola máquina virtual de Java (JVM). Proporciona un modelo extenso de seguridad de modo que los componentes puedan funcionar en un ambiente seguro. Sin embargo, con los permisos apropiados, los

componentes pueden ser reutilizados y cooperar, a diferencia de otros entornos de aplicación de Java. El framework de OSGi proporciona un conjunto extenso de mecanismos para hacer posibilitar esta cooperación de forma segura.

La presencia de un middleware basado en OSGi en muchas diversas industrias está creando un gran mercado de software para componentes software de OSGi. La definición rígida de la plataforma OSGi permite que los componentes que pueden funcionar en una gran variedad.

La adopción de las especificaciones de OSGi puede, por lo tanto, reducir costes de desarrollo de software y proporciona también nuevas perspectivas.

3. Método dirigido por modelos aplicado en el caso de estudio

El método que se aplica en el caso de estudio desarrollado, presentado en [3], aplica las guías definidas por la Model Driven Architecture (MDA), propuesta por el Object Managemente Group (OMG), y las Software Factories, propuestas por Microsoft. El método proporciona (1) un lenguaje de modelado (PervML) para especificar sistemas pervasivos utilizando primitivas conceptuales adecuadas para este dominio, (2) un framework de implementación que proporciona una arquitectura común para todos los sistemas desarrollados siguiendo este método y (3) un motor de transformación que convierte las especificaciones PervML en código que extiende el framework.

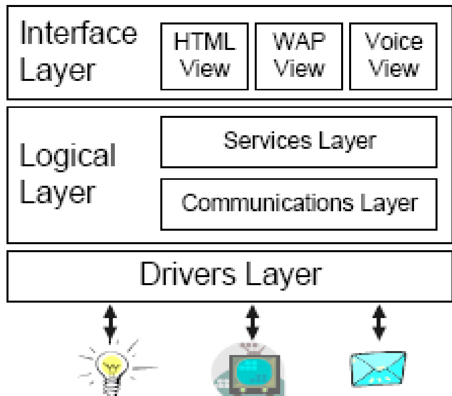


Figura 3. Arquitectura del sistema

El framework de implementación (descrito detalladamente en la Sección 5), sigue un estilo arquitectónico estructurado en capas (Figura 3) y ha sido implementado utilizando la tecnología para pasarelas residenciales OSGi. La integración de los distintos dispositivos y sistemas software que componen el sistema se realiza mediante drivers. Por otra parte, se ha aplicado el patrón Model-View-Controller [4], de manera que es posible acceder al sistema desde múltiples interfaces (vistas) a través de la funcionalidad proporcionada por un elemento controlador. La Figura 4 muestra los pasos que define el método. Las burbujas sombreadas corresponden a pasos automatizados.

1. El analista del sistema especifica los requisitos utilizando el lenguaje de modelado PervML. Mediante tres modelos el analista describe (1) los tipos de servicios disponibles en el sistema, (2) la cantidad de estos servicios que se encuentran disponibles en cada ubicación del sistema y (3) como estos interactúan entre sí al cumplirse alguna condición.

Figura 4. Pasos para aplicar el método propuesto

2. El arquitecto del sistema selecciona el tipo y cantidad de dispositivos o sistemas software que son más adecuados para proporcionar los servicios descritos por el analista. La selección puede atender a razones económicas o a las características del entorno, por ejemplo. Para ello, el arquitecto del sistema utiliza otros tres modelos de PervML en los que describe (1) el tipo de dispositivo o sistemas software que se utilizan, (2) los elementos que implementan cada servicio y (3) las acciones que se llevarán a cabo para proporcionar cada operación de los servicios.

3. Un desarrollador implementa en OSGi los drivers para gestionar los dispositivos o sistemas software que han sido seleccionados por el arquitecto. Estos drivers permiten acceder desde el framework a la funcionalidad proporcionada por los dispositivos o sistemas software externos. Deben ser desarrollados manualmente, ya que han de lidiar con cuestiones muy específicas de tecnología y fabricante. Es posible reutilizar drivers desarrollados en proyectos anteriores si se utiliza el mismo tipo de dispositivo.

4. Se aplica el motor de transformación a la especificación PervML. Como resultado de este paso se generan automáticamente archivos Java y otros recursos (archivos Manifest, etc.)

5. Los archivos Java son configurados para utilizar los drivers seleccionados. Este paso consiste en indicar los identificadores OSGi de los drivers.

6. Finalmente, los archivos generados son compilados, empaquetados en bundles (archivos JAR) e instalados en el servidor OSGi junto con el framework y los drivers.

El método está centrado en el desarrollo del software del sistema pervasivo. La definición de procedimientos para la instalación física de los dispositivos, redes, etc. no están incluidos en la descripción del método.

4. PervML: Modelado conceptual de sistemas pervasivos

PervML (Pervasive Modeling Language) [5] es un lenguaje de dominio específico para describir sistema pervasivos de un modo independiente de la tecnología. PervML promueve la separación de roles, de modo que los desarrolladores pueden categorizarse en analistas y arquitectos.

Los Analistas del Sistema capturan los requisitos y describen el sistema pervasivo a alto nivel de abstracción utilizando la metáfora servicio como la principal primitiva conceptual para abstraer la funcionalidad que requiere el usuario. Los analistas construyen tres modelos gráficos que constituyen lo que llamamos la Vista del Analista. En estos modelos el analista describe (1) los tipos de servicios (mediante sus interfaces, las relaciones entre ellos y un diagrama de transición de estados para describir su comportamiento), (2) los distintos componentes que van a proporcionar los servicios del sistema y (3) como estos componentes interactúan entre sí.

Por otra parte, los Arquitectos del Sistema especifican qué dispositivos y/o sistemas software externos implementarán los servicios. Se denomina proveedor de enlace (binding provider) a aquellos elementos que son responsables de enlazar el sistema pervasivo con su entorno físico o lógico. Por ejemplo, un sensor de luminosidad se encarga de medir una característica física del entorno, mientras que un servidor de correo electrónico permite enviar información a agente que están fuera del ámbito del sistema. Los arquitectos construyen otros tres modelos que constituyen lo que se denomina la Vista del Arquitecto. En ellos el arquitecto describe (1) cada tipo de proveedor de enlace (mediante su interfaz), (2) los proveedores de enlace que se utilizan en cada componente, y (3) las acciones que deben llevarse a cabo cuando se invoca cada operación de cada componente.

4.1. Vista del Analista

El objetivo del analista del sistema pervasivo es capturar los requisitos del sistema utilizando conceptos de alto nivel de abstracción. En la Vista del Analista de PervML el analista describe el sistema pervasivo utilizando la metáfora servicio como la principal primitiva conceptual. Esta vista está compuesta por tres modelos, que se describen a continuación. En la figura 5 se muestran ejemplos de dichos modelos.

El **Modelo de Servicios** se utiliza para describir los distintos tipos de servicios disponibles en el sistema. En un sistema PervML puede haber distintas instancias de un mismo tipo de servicio, por lo que este modelo se utiliza para especificar las características comunes que comparten dichas instancias.

Para modelar un servicio es necesario describir sus operaciones y sus relaciones, por tanto, se representará como una clase de UML.

La signatura de las operaciones de cada servicio quedará definida en la clase que lo represente. La semántica de cada operación se describirá mediante las pre y postcondiciones utilizando Object Constraint Language (OCL).

Las relaciones expresadas en el diagrama de clases serán de dos tipos: de generalización/especialización y de agregación. Una relación de generalización/especialización es una relación taxonómica entre un elemento más general y uno más específico. Es la relación que existe, por ejemplo, entre un servicio

de iluminación y un servicio de iluminación con graduación de la intensidad. Será necesario proporcionarles una semántica bien definida utilizando un marco como el propuesto en [6]. Por otra parte, las relaciones de agregación se utilizarán para modelar aquellos servicios que, conceptualmente, suponen la agrupación de otros servicios. Por ejemplo, un servicio que proporciona gestión de la iluminación mediante detección de presencia necesitará para su funcionamiento de servicios de iluminación y de servicios de detección de presencia. Como en el caso de la generalización, será necesario proporcionarles una semántica bien definida utilizando un marco como el propuesto en [7]. Para describir el comportamiento del servicio se utilizará un Diagrama de Transición de Estados (DTE). Mediante este DTE se mostrará la secuencia válida de operaciones que puede ocurrir en la vida de un servicio. Las transiciones entre estados tendrán asociada la operación que desencadena el cambio de estado y, si es necesario, una restricción, o guarda, que debe satisfacerse antes de realizar la transición.

Un servicio también puede tener conceptualmente un papel activo; por ejemplo, todo servicio de regulación de temperatura deberá encenderse cuando la temperatura de la ubicación en la que se encuentra descienda del umbral que tiene establecido. Para poder expresar este comportamiento la descripción de los servicios incluirá disparadores expresados en OCL.

En el **Modelo Estructural** se especifican las instancias de cada tipo de servicio que hay en el sistema. Se representan mediante un diagrama de componentes de UML 2.0. La interfaz que implementa el componente indica que tipo de servicio implementa. Se establecen relaciones de dependencia entre componentes para indicar que un componente utiliza la funcionalidad proporcionada por otro. Las relaciones entre componentes son determinadas en el modelo de servicios. En nuestro caso el sistema de seguridad requiere de la funcionalidad de un servicio de detección de presencia y otro de grabación.

El **Modelo de Interacción** se utiliza para describir la comunicación producida como reacción a un evento del sistema. Se representa mediante diagramas de interacción UML 2.0. La condición de disparo de la interacción se representa usando OCL. Las acciones descritas en el diagrama deben ser ejecutadas cuando la condición se satisfaga. Por ejemplo una interacción podría ser la reducción de la intensidad de la iluminación cuando alguien se acerca a la pantalla. Las acciones especificadas en una interacción no son ejecutadas como respuesta explícita del usuario, sino que son invocadas como reacción a una situación. Por tanto, este puede ser un mecanismo adecuado para especificar funcionalidades dependientes del contexto.

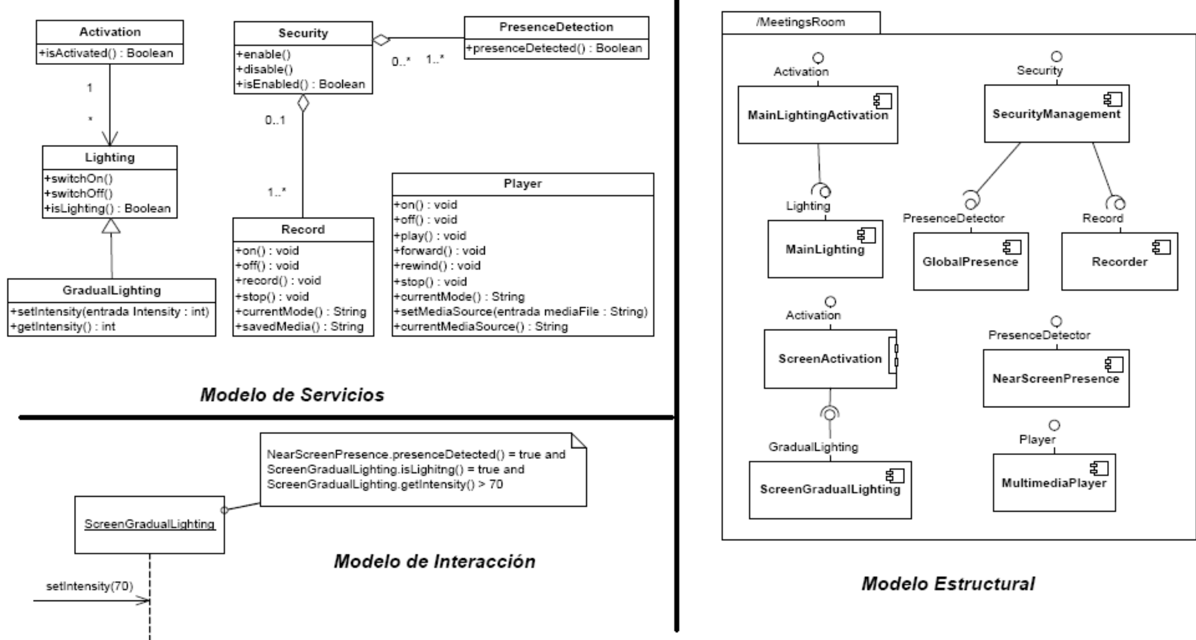


Figura 5. Ejemplo de modelado de la vista del analista

4.2. Vista del Arquitecto

Para tener una descripción completa de un sistema pervasivo es necesario indicar qué dispositivos y sistemas software externos implementarán los servicios del sistema. Llamamos a estos elementos proveedores de enlace (binding providers) ya que enlazan el sistema pervasivo con su entorno físico o lógico. En la Vista del Arquitecto de PervML, el arquitecto especifica qué tipos de dispositivos y sistemas software son usados en el sistema, qué elementos implementa cada servicio y como estos elementos proporcionan esos servicios. Describimos a continuación los modelos que debe desarrollar el arquitecto. Se incluyen ejemplos en la figura 6.

El **Modelo de Proveedores de Enlace** se utiliza para describir los distintos tipos de dispositivos y sistemas software que van a interactuar con el entorno. Sin estos elementos, el sistema no podría realizar la funcionalidad especificada. Las interfaces de los proveedores de enlace se especifican en un diagrama de clases.

Este modelo describe la funcionalidad dada para un tipo de dispositivo o sistema software, pero no queda enlazado específicamente con ningún elemento final.

En **la Especificación Estructural** de los Componentes el arquitecto indica qué proveedores de enlace van a implementar cada servicio del sistema. Se debe resaltar que un mismo proveedor de enlace puede ser usado para proporcionar diversos servicios. Por ejemplo, se puede utilizar una cámara con capacidad de detectar presencia para implementar dos servicios distintos: uno de detección de presencia y otro de grabación de video.

Finalmente, **la Especificación Funcional** de los Componentes indica las acciones que se llevarán a cabo cuando se invoque cada una de las operaciones de cada uno de

los servicios. Para ello se utiliza el Abstract Semantics Language (ASL). En el ejemplo se muestra cómo se implementa una operación de un servicio de detección de presencia. El servicio debe devolver un valor booleano, mientras que el dispositivo indica la probabilidad de que realmente haya alguien en la sala, por lo que es necesario fijar un umbral (en este caso, 80 %).

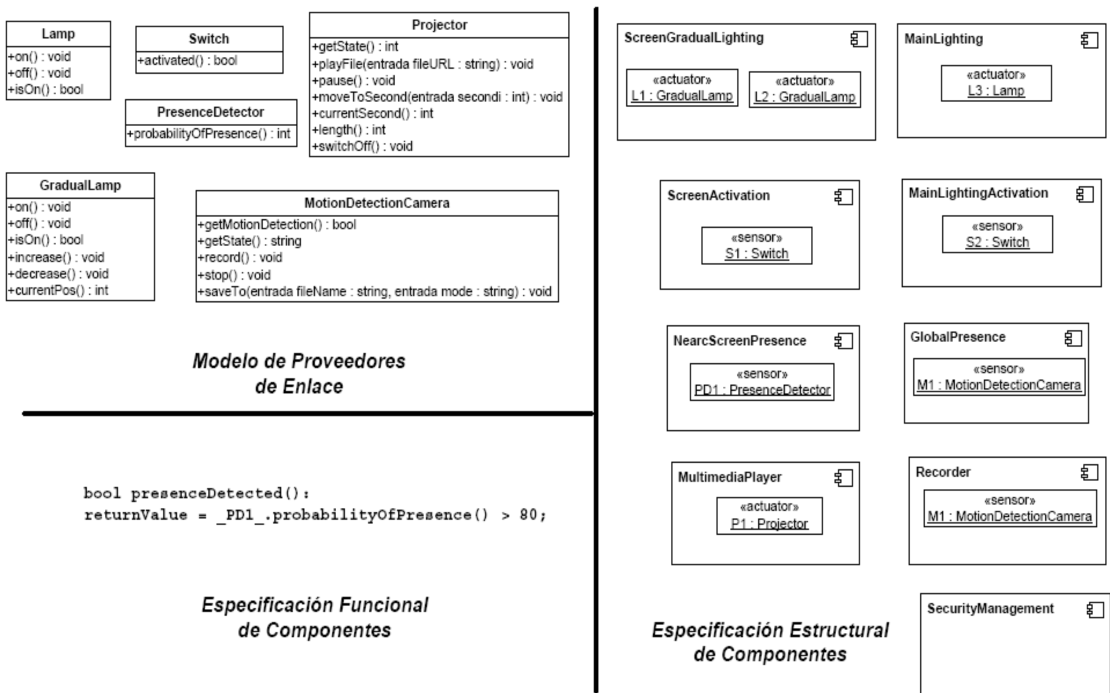


Figura 6. Ejemplos de los modelos para la vista del arquitecto

5. Framework de Soporte a PervML

El desarrollo de un sistema pervasivo supone el uso de diversas tecnologías para satisfacer los requisitos de los usuarios. Habitualmente estas tecnologías proporcionan a los desarrolladores constructores de bajo nivel de abstracción. MDA [8] y las Factorías de Software [9] proporcionan estrategias para aumentar el nivel de abstracción y la productividad en el desarrollo de sistemas complejos.

Para salvar el salto de abstracción entre los lenguajes de modelado y las tecnologías de implementación, el enfoque de las Factorías de Software propone la construcción de frameworks, tal y como se muestra en la Figura 7 (A). Aplicando principios de la ingeniería de dominios, se desarrolla un framework que aumenta el nivel de abstracción de la tecnología. De esta manera se reduce la cantidad de código que debe ser generada a partir de los modelos de alto nivel de abstracción.

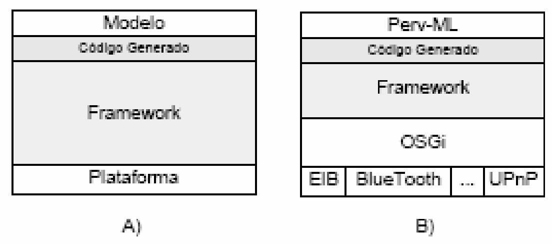


Figura 7. Estrategia propuesta por las Factorías Software y nuestra aplicación.

Se describe a continuación el framework [10] que da soporte al lenguaje de modelado PervML. El desarrollo de este framework de implementación se ha realizado de manera conjunta con Carlos Cetina, ya que su Proyecto Final de Carrera (II-B-DSIC-112/05) se ha realizado en un contexto similar. El framework, implementado utilizando el middleware OSGi, proporciona una serie de clases abstractas que deberán ser convenientemente extendidas para obtener una aplicación funcionalmente operativa. Las primitivas que proporciona el framework son similares a las que utiliza en el lenguaje de modelado.

El framework para sistemas pervasivos ha sido diseñado con la intención de servir de soporte a los sistemas especificados utilizando el lenguaje de modelado PervML. Por lo tanto, un objetivo crítico ha sido proporcionar primitivas similares a aquellas utilizadas por el lenguaje. Este requisito ha hecho que la arquitectura de los sistemas generados por el framework siga una estructura y estrategia de ejecución análoga a la propuesta por el lenguaje de modelado para especificar los sistemas. A continuación se describe tanto la estructura como la estrategia global de ejecución, y por último el uso del framework.

5.1. Estructura del Framework

La estructura de los sistemas desarrollados con el framework para sistemas pervasivos sigue los patrones arquitectónicos Layers y Model-View-Controller. Mediante el patrón Layers, organizamos los elementos del sistema en capas con unas responsabilidades bien definidas. Por otra parte, mediante el patrón Model-View-

Controller proporcionamos soporte a la existencia de varias interfaces de usuario para interactuar con el sistema.

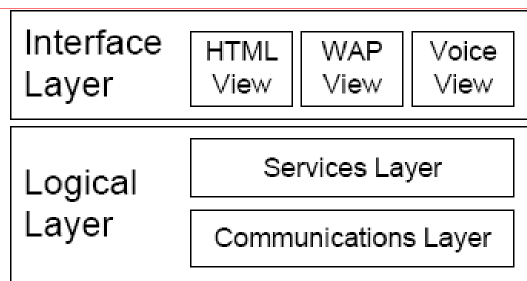


Figura 8. Estructura del framework

El framework se puede estructurar en dos partes bien diferenciadas como se muestra en la figura 8: el soporte a los elementos de la Capa Lógica (proveedores de enlace, componente e interacciones) y el soporte a las Interfaces de Usuario que se encuentra en la Capa de Interfaz.

5.1.1. Capa Lógica

La capa lógica se puede dividir en dos capas: la capa de Comunicaciones y la capa de Servicios.

La Capa de Comunicaciones proporciona una representación de los proveedores de enlace (binding providers) que son usados en capas superiores. Esta capa contiene los aspectos de la comunicación con los dispositivos y sistemas software que son independientes de tecnología y fabricante. Por ejemplo, se encarga de registrar los accesos al proveedor de enlace, de notificar los cambios en el driver a todos los elementos de capas superiores que requieran esa información, etc. Por todo ello, existe una relación uno a uno entre los drivers y los elementos de esta capa. Por ejemplo, si el sistema contiene un sensor de temperatura de la tecnología LonWorks, existirá un driver para tratar con las cuestiones específicas de esa tecnología y un proveedor de enlace con funcionalidad común a todos los sensores de temperatura.

La Capa de Servicios se encarga de abstraer la funcionalidad que implementan las capas inferiores para ofrecerla tal y como la esperan los usuarios del sistema. Cada uno de los elementos de esta capa, a los que llamamos componentes, proporciona un tipo de servicio. Para hacer uso de la funcionalidad proporcionada por un tipo de servicio se establece un contrato (definido mediante pre y post condiciones de las operaciones) y un protocolo, que establece las operaciones que pueden invocarse en un momento dado. Para implementar su funcionalidad, un componente puede hacer uso de proveedores de enlace (dispositivos y sistemas software externos) o de otros componentes.

Finalmente, en esta capa también se pueden implementar interacciones entre componentes. En este ámbito entendemos una interacción como una secuencia de comunicaciones entre componentes, conceptualmente no relacionados a priori, para proporcionar una funcionalidad requerida por el usuario. Por ejemplo, el usuario puede requerir que se atenúe la iluminación de una sala al iniciarse la reproducción de un objeto multimedia. Conceptualmente, los servicios de iluminación y de reproducción de objetos multimedia no están relacionados entre sí pero, en un sistema en concreto, el usuario puede desear establecer una interacción entre ellos.

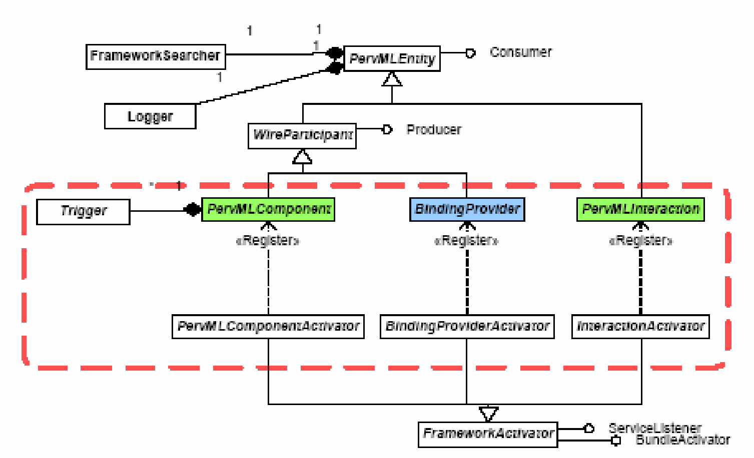


Figura 9. Estructura de clases para soportar las capas de Comunicaciones y Servicios

La Figura 9 muestra la estructura de clases que proporcionan soporte a la Capa Lógica. Los elementos dentro del recuadro punteado serán los utilizados finalmente para extender del framework y generar el sistema final, tal y como se describirá en la Sección 5.3.

Las clases PervMLEntity y WireParticipant se encargan de implementar las interfaces que permiten utilizar el mecanismos de Wire de OSGi, así como de facilitar acceso a objetos de las clases FrameworkSearcher y Logger. El resto de clases de esa jerarquía (en el recuadro) contienen los puntos de extensión del framework. En ellas se definen los atributos que deben inicializarse al instanciar el framework, y se implementan, utilizando el patrón de diseño Template Method, las estrategias de ejecución de cada uno de los elementos.

La otra jerarquía de clases mostrada en la figura 9 contiene los elementos de soporte a la creación de los activadores. FrameworkActivator es la clase más abstracta y se encarga de definir la estrategia general de los activadores y los puntos de extensión. También implementa algunas operaciones que se encargan de realizar funcionalidad atómica, como registrar en OSGi un objeto bajo una interfaz o crear un Wire entre dos servicios. El resto de activadores más específicos implementan algunos de los puntos de extensión, dejando otros abstractos para que sean escritos al instanciar el framework.

5.1.2. Capa de interfaz

La Capa de Interfaz se encarga de proporcionar el acceso al sistema por parte de cualquier tipo de usuario (tanto personas como otros sistemas software). En un contexto pervasivo es habitual proporcionar distintos tipos de interfaces para acceder al mismo sistema. Por ejemplo, una interfaz web para acceder remotamente, una PDA o un teléfono móvil para acceder con movilidad, o una interfaz a través de TV para gestionar de manera centralizada un hogar. Por ello, se ha considerado adecuado aplicar el patrón Model-View-Controller en el framework. Siguiendo este patrón, sobre la capa de servicios se sitúa un elemento controlador. Las distintas

interfaces (vistas) del sistema se comunican con este controlador para interactuar con el sistema.

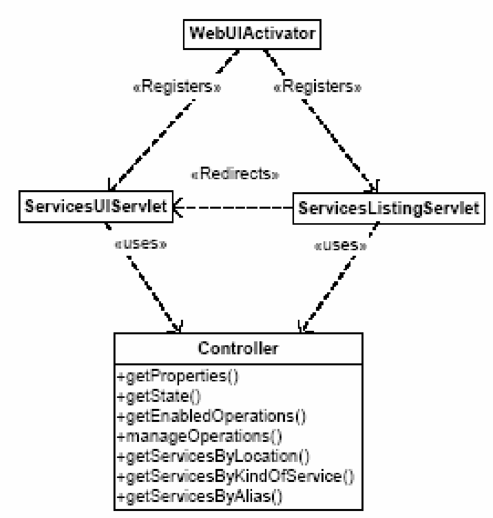


Figura 10. Estructura de clases para soportar la capa de Interfaz

La figura 10 muestra el diagrama de clases que proporcionan soporte a la Capa de la Interfaz. Como ya se ha comentado al describir la arquitectura, se ha aplicado el patrón MVC. En nuestros sistemas, los elementos que forman parte del Modelo son los Component; mientras que la capa de interfaz contiene el Controlador y las distintas Vistas.

Al realizar el diseño del Controlador se ha considerado que el usuario realizará dos tareas a la hora de interactuar con el sistema:

1. Seleccionar el servicio con el que desea interactuar de entre todos los disponibles en el sistema. Para ello será necesario proporcionar mecanismos de estructuración, acceso y búsqueda de los componentes.

2. Interactuar con un servicio en concreto. En esta interacción el usuario recibirá información del servicio y podrá solicitar la ejecución de alguna funcionalidad que este ofrece.

Siguiendo esta filosofía, el controlador tiene dos grupos de métodos para soportar estas tareas:

- El primero de los grupos está formado por los métodos getServicesByLocation, getServicesByKindOfService y getServicesByAlias. Estos métodos devuelven una lista con información sobre los servicios del sistema, filtrándolos por distintos criterios.

- El segundo de los grupos está formado por los métodos getProperties, getState, getEnabledOperations y manageOperation. Las tres primeras operaciones son invocadas a la hora de crear la interfaz de usuario, ya que devuelven información de un componente específico. En concreto, la primera de ellas devuelve las propiedades invariables de un servicio (el tipo de servicio, su ubicación, etc.), la segunda devuelve el estado del servicio (por

ejemplo, si se encuentra encendido o apagado, o si está detectando presencia) y, finalmente, la tercera devuelve las operaciones que el componente puede ejecutar en ese instante. Por último, manageOperation se encarga de invocar una operación en un Component utilizando las capacidades de reflexión de Java.

En la figura 10 también se muestran las clases que proporcionan la vista a través de páginas web. Se trata de dos servlets de Java, cada uno de los cuales se utiliza para soportar una de las dos tareas descritas anteriormente. Los servlets realizan invocaciones a las operaciones del controlador para generar las páginas web que serán mostradas a los usuarios.

Es importante hacer notar que todas las clases de esta capa son concretas; es decir, no necesitan extenderse ya que son válidas para todos los sistemas. Esto se ha conseguido gracias a que (1) todos los componentes implementan una interfaz que puede ser utilizada por el controlador y (2) se han utilizado las capacidades de reflexión de Java a la hora de ejecutar las operaciones específicas de los componentes.

5.2. Estrategia Global de Ejecución

Una vez presentada la estructura de los sistemas desarrollados utilizando el framework, es necesario describir su estrategia de ejecución; es decir, las reglas que definen las secuencias de acciones que suceden en el sistema cuando este se encuentra en funcionamiento.

5.2.1. Secuencia de acciones

Existen dos posibles modos de iniciar una secuencia de acciones:

1. Un *Driver* notifica a un *BindingProvider* de un cambio en el entorno.

2. El usuario solicita, utilizando una UI, la ejecución de una funcionalidad de un servicio.

A continuación se describen los pasos que se llevan a cabo al realizar estas dos secuencias de acciones; así como la estrategia de ejecución de las operaciones de los servicios y de las interacciones. Finalmente, se describe brevemente el funcionamiento de la interfaz web que se ha desarrollado.

Cuando un *Driver* notifica a un *BindingProvider* que ha habido un cambio en el entorno, el *BindingProvider* notificará este hecho a los *Component* que hacen uso de él. Estos, a su vez, evaluarán sus disparadores (*Triggers*) y notificarán a aquellos otros *Component* o *Interaction* en cuyas condiciones de disparo participa.

Detalladamente, y como indica gráficamente la figura 11, se realizan los siguientes pasos:

1. El *Driver* notifica al *BindingProvider* que ha sucedido un cambio en el entorno. No indica de qué cambio se trata.

2. El *BindingProvider* notifica a todos los elementos *Component* que lo utilizan de un cambio en el entorno. No indica de qué cambio se trata.

3. El *Component* evalúa las condiciones de sus disparadores. Para ello podrá realizar invocaciones sobre las operaciones de los *BindingProviders* que utiliza (entre ellos quizá el que inició las acciones) o de otros *Component* con los que se relaciona.

4. El Component consulta los resultados de aquellas de sus operaciones que devuelven algún valor (aquellas operaciones como "getIntensity(): int" o "isLighting(): boolean").

    a. Si ninguno de los valores devueltos ha cambiado desde la última vez que los consultamos:

        i. Finaliza la secuencia de acciones

    b. Si algún valor ha cambiado:

        i. Almacenamos los nuevos valores.

        ii. Continuamos con los siguientes pasos

5. El Component notifica de que ha habido un cambio en el resultado de alguna de sus operaciones a aquellos Component e Interaction que le escuchan.

6. Los elementos notificados (*Component* o *Interaction*) evalúan las condiciones de sus disparadores (el *Interaction* sólo tiene una condición), para lo cual invocarán alguna de las operaciones del *Component* que realizó la notificación.

7. Ocasionalmente, como resultado de los pasos 3 o 6, se inicia la ejecución de alguna operación de un *Component* o las acciones de una *Interaction*.

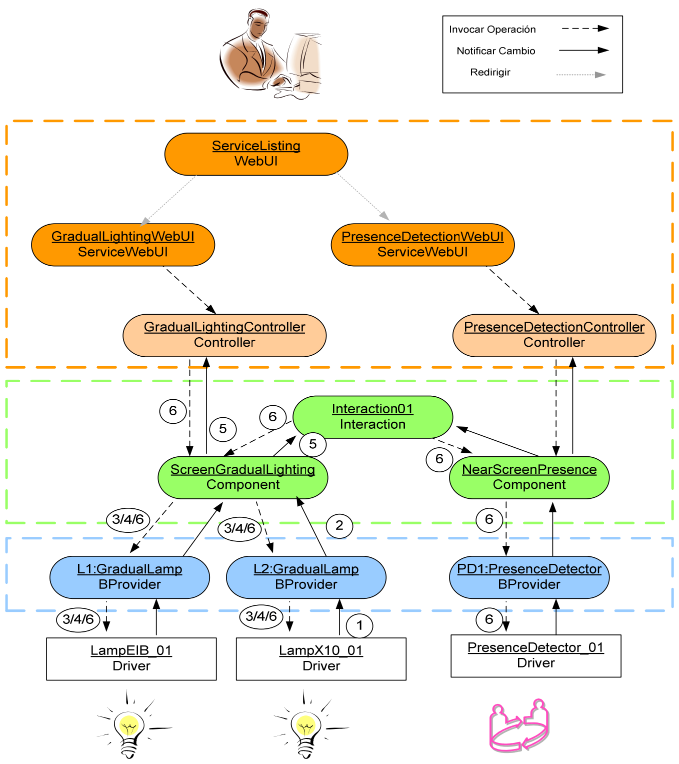


Figura 11. Pasos que se realizan ante un cambio en el entorno

Cuando un usuario desea que se lleve a cabo la funcionalidad proporcionada por un servicio, en primer lugar debe buscar en la UI la instancia del servicio (*Component*) con la que quiere interactuar. Entonces invoca la operación (pulsa un botón, pasa el puntero del ratón por una región, dice un comando, etc.) y la UI solicita a un *Controller* que dicha operación se lleve a cabo. El *Controller*, si es posible, invoca la operación sobre el *Component*.

Detalladamente, y como indica gráficamente la figura 12, se realizan los siguientes pasos:

1. El *Usuario* selecciona en la UI la instancia del servicio que quiere manejar. En el sistema web, esto se realiza mediante varias páginas que permiten ver los servicios ordenados por tipo o por ubicación; o mediante búsquedas por nombre.

2. La *UI* transmite al usuario información sobre el componente seleccionado, así como aquellas operaciones que puede ejecutar. Para ello se realizan los siguientes pasos:

   a. La *UI* solicita la información al *Controller* del tipo de servicio con el que desea interactuar. Esta información está estructurada en tres paquetes:

i. Información general del *Component*: Actualmente se muestra el tipo de servicio de que se trata y su ubicación.

ii. Información sobre la situación actual del *Component*: Que viene determinada por los resultados de aquellas de sus operaciones que devuelven un valor y no reciben parámetros.

iii. Operaciones que puede invocar el usuario en ese instante.

b. El *Controller* interactúa con el *Component* para obtener la información necesaria, para lo cual:

i. Obtiene la información estática relativa al componente. Esta información consiste en localización, tipo y nombre.

ii. Invoca aquellas operaciones del *Component* que devuelven un valor y no reciben parámetros. Los resultados los devuelve empaquetados en un diccionario.

iii. Consulta al Component las operaciones que puede ejecutar en ese instante, y para cada una de las operaciones el nombre y el tipo de sus parámetros. El resultado lo devuelve en un diccionario en el que el campo clave es el nombre de la operación y el campo valor es otro diccionario, en el que la clave es el nombre del parámetro y el valor es el tipo del mismo.

c. La *UI* transmite al *Usuario* la información devuelta por el Controller. El modo en que se transmite esta información es dependiente del tipo de UI (página Web, PDA, secuencias de voz, imagen del entorno, etc).

3. El *Usuario* ordena la ejecución de una operación (pulsa un botón, pasa el puntero del ratón por una región, dice un comando, etc.). Si es necesario, la *UI* se encarga de recoger los parámetros para iniciar la petición de ejecución.

4. La *UI* transmite la petición de ejecución al *Controller*, indicándole el *Component* sobre el que quiere invocar la operación, el nombre de la operación, y los parámetros de esta, si los necesita.

5. El Controller invoca la operación en el Component.

6. El Controller vuelve a interactuar con el componente para obtener la información descrita en el punto 2 b

a. Si la operación se ha ejecutado de forma correcta se devuelve la información que refleja el estado actual a la UI.

b. Si ha habido algún problema durante la ejecución de la operación, junto a la información que refleja el estado actual del componente, el controller devuelve a la UI un diccionario que contiene los errores sucedidos y el instante en el que se dieron.

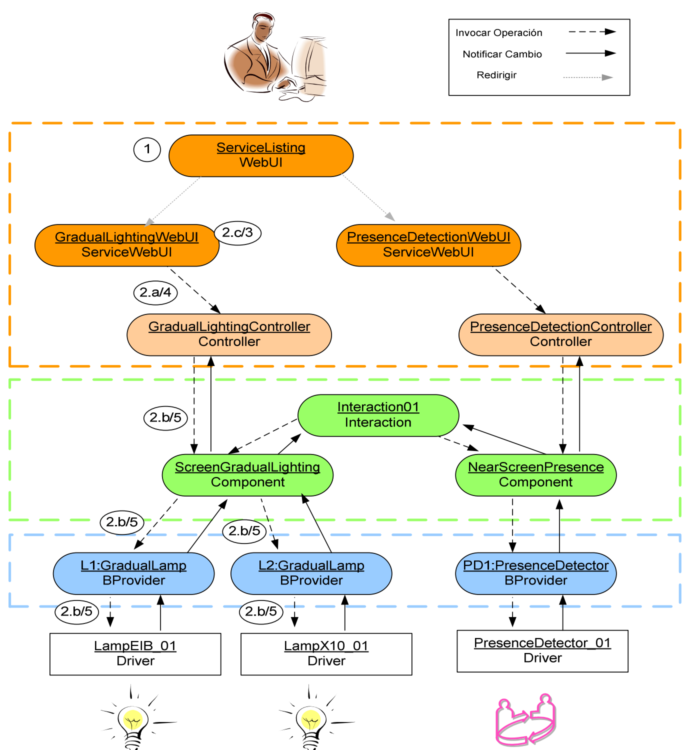


Figura 12. Pasos ante la solicitud de un servicio por parte del usuario

5.2.2. Ejecución de operaciones

La estrategia de ejecución de operaciones es una de las partes más importante de la estrategia de ejecución general. Para ejecutar una operación de un *Component* se realizan los siguientes pasos:

1. Comprobar que es posible ejecutar la operación. Para ello será necesario:

a. Comprobar que, desde el estado actual del DTE, está permitido llevar a cabo la operación. A su vez, este paso se puede descomponer en:

i. Comprobar que realmente el *Component* se encuentra en el estado que está almacenado. Para ello se evaluará la fórmula asociada a ese estado.

1. Si no se cumple la fórmula:

a. Evaluar las fórmulas de todos los estados del DTE

i. Si sólo se cumple una: se actualiza el estado actual

ii. Si se cumplen varias: se elige un estado al azar

iii. Si no se cumple ninguna: Registrar el error y se detiene la ejecución

ii. Comprobar que la operación se encuentra en la lista de operaciones válidas en el estado actual.

1. Si no es encuentra en la lista: se registra el error y se detiene la ejecución.

b. Comprobar la precondición. Para ello se evaluará la fórmula asociada.

i. Si la fórmula *no* se satisface: se registra el error y se detiene la ejecución.

2. Deshabilitar la recepción de notificaciones. De este modo la ejecución de las operaciones tiene un carácter atómico. Esto se realiza para evitar comportamiento no deseado y posibles ciclos como consecuencia de la invocación de operaciones en los *BindingProviders*, los cuales a su vez podrían enviar notificaciones al propio *Component*. (Por ejemplo, un servicio de Iluminación que debe activar varias bombillas. Al activar una bombilla, esta notificará al *Component* de un cambio en su estado; lo cual desencadenará una serie de acciones en él (comprobación de disparadores, notificación del *Component* a otros elementos, etc.) antes de que se hayan activado el resto de bombillas).

3. Realizar las acciones de la operación. Para ello se sigue la siguiente estrategia:

a. Conseguir las referencias de los *Component* o *BindingProviders* que se utilicen en los siguientes pasos.

i. Si no se consiguen todas las referencias: se registra el error y se detiene la ejecución

b. Ejecutar las acciones que implementan la operación.

i. Si hay alguna excepción: se registra el error y se detiene la ejecución

ii. Si hay que devolver un valor, se almacena en la variable *returnValue*.

c. Si hay que devolver un valor: hacer *return* de la variable *returnValue*.

4. Activar la recepción de notificaciones.

5. Comprobar que si la operación ha tenido el efecto esperado. Para ello se evaluará la fórmula de la postcondición.

a. Si la fórmula se satisface:

i. Transitar en el DTE

b. Si la fórmula *no* se satisface:

i. Registrar el error.

ii. Averiguar el estado del DTE en el que se encuentra el *Component*.

1. Evaluar las fórmulas de todos los estados del DTE

a. Si sólo se cumple una: se actualiza el estado actual

b. Si se cumplen varias: se elige un estado al azar

c. Si no se cumple ninguna: Registrar el error y se detiene la ejecución

6. Evaluar los disparadores del *Component*, por si se ha perdido alguna notificación durante el tiempo que se deshabilitaron las notificaciones.

7. Para todas las operaciones que no definen el estado, el Component consulta los resultados de aquellas de sus operaciones que devuelven algún valor (aquellas operaciones como "*getIntensity(): int*" o "*isLighting(): bolean*").

a. Si algún valor ha cambiado:

i. Almacenamos los nuevos valores.

ii. Notificamos del cambio a los *Component* o *Interaction* suscritos.

5.3. Instanciación del Framework

En esta sección se describe cómo instanciar el framework para implementar un sistema pervasivo. En cada subsección se proporciona una breve visión para cada una de los elementos del sistema.

5.3.1. Creación de los Proveedores de Enlace

Como puede observar en la figura 13, tanto los drivers, como los proveedores de enlace con los que se relacionan, implementan una misma interfaz. Por ejemplo, los dos drivers para acceder a lámparas que proporcionan iluminación gradual y su proveedor de enlace implementan la interfaz GradualLamp.

Sólo existe un proveedor de enlace por cada tipo. Será en el activador donde se creen dos instancias, las cuales recibirán como parámetros de su constructor su propio identificador y el identificador del driver con el que están emparejados. A continuación se muestra el código contenido en el punto de extensión del activador (método specificStart), donde los identificadores de los drivers son LampEIB_01 y LampX10_01.

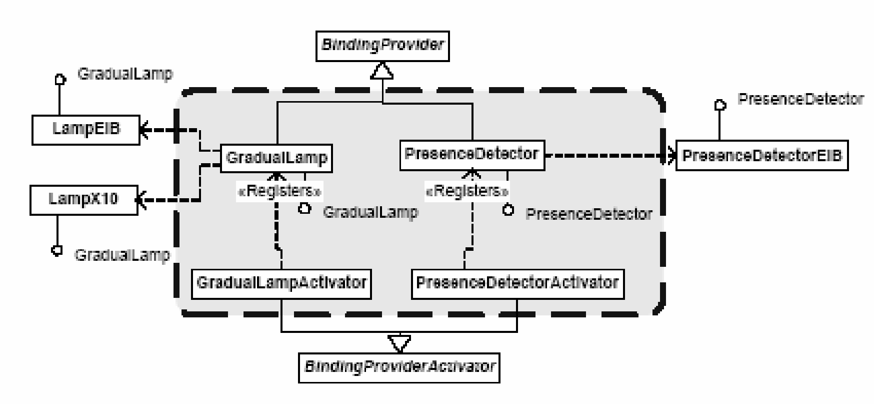


Figura 13. Estructura de clases para implementar los proveedores de enlace

```
public void specificStart(){
registerBProviders("LampEIB_01","L1",new BProvider("L1",this.context));
registerBProviders("LampX10_01","L2",new BProvider("L2",this.context));
}
```

La implementación de las operaciones del proveedor de enlace se limita a realizar la invocación en el driver. Para ello, se hace uso de un método llamado runMethod que realiza la invocación utilizando las capacidades de reflexión de Java. A continuación se muestra el código de la operación On() del proveedor de enlace que representa una lámpara gradual.

```
public void On(){
```

```
runMethod("On",new Object[0]);
}
```

5.3.2. Creación de los Componentes e Interacciones

La figura 14 muestra la estructura de clases para implementar los componentes del sistema. Como puede observarse cada componente está implementado en dos clases (por ejemplo, GradualLighting y ScreenGradualLighting), atendiendo al siguiente criterio:

- La clase más abstracta (GradualLighting en el ejemplo) implementa aquella funcionalidad que es común para todos los componentes que proporcionan un mismo servicio. Esto incluye métodos para calcular las operaciones que pueden ser invocadas en un momento dado, las pre y post condiciones de las operaciones y la estrategia general de ejecución de las operaciones del servicio.

- La clase más específica (ScreenGradualLighting en el ejemplo) implementa las acciones de las operaciones. Hay que tener en cuenta que en el sistema podría existir otro componente que proporcionase el servicio de iluminación gradual, el cual utilizase otro tipo de dispositivos (por ejemplo, lámparas que permiten indicar un porcentaje de luminosidad o lámparas que sólo permiten 4 posiciones de intensidad) y, en ese caso, las acciones a llevar cabo cuando se ejecutasen las operaciones serían necesariamente distintas aunque el servicio proporcionado fuese el mismo.

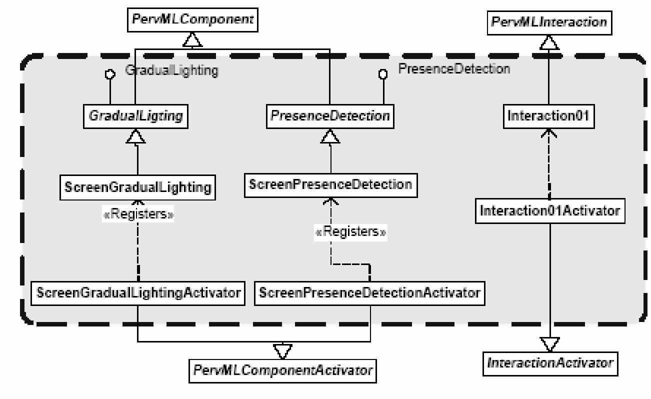


Figura 14. Estructura de clases para implementar los componentes y las interacciones

A continuación se muestra la implementación del método para fijar la intensidad del componente que proporciona el servicio GradualLighting. La estrategia que se sigue consiste en dividir la intensidad entre las dos lámparas, de manera que el primer 50% lo proporciona la lámpara L1 y el segundo 50% L2.

```
public void setIntensityImplementation(int intensity){
    org.oomethod.interfacesBProviders.gradualLamp.Interface L1, L2;
    L1 =(org.oomethod.interfacesBProviders.gradualLamp.Interface)
            frameworkSearcher.getBProvider("L1");
    L2 =(org.oomethod.interfacesBProviders.gradualLamp.Interface)
            frameworkSearcher.getBProvider("L2");
    if ( intensity >= 50) {
            L1.on();
            L2.set( (intensity - 50)*2 );
    } else {
            L1.set( intensity *2 );
            L2.off();
            }
    }
```

Por otra parte, el activador del componente es el encargado de inicializar algunas variables específicas y de crear los canales de comunicación (Wires) con los proveedores de enlace. Para ello implementa las operaciones que forman los puntos de extensión proporcionados por el framework.

```
package org.oomethod.components.screenGradualLighting;
public class Activator extends ComponentActivator {
    protected void initialize(){
            componentPID="ScreenGradualLighting";
            implLocation="/Dept/Floor1/Lab103";
            componentInstance=new Component(this.context,componentPID);
    }
    protected void createAllWires(WireAdmin wa){
            this.createWire(wa,"L1",componentPID);
            this.createWire(wa,"L2",componentPID);
    }
  }
```

En la figura 14 también aparecen las clases para implementar la Interacción del ejemplo. Una interacción deberá implementar los puntos de extensión definidos por el framework:

- el método checkCondition evalúa la condición de inicio de la interacción.
- el método doAction realiza las invocaciones sobre los distintos componentes que participan en la interacción.

A modo de ejemplo, a continuación se muestra el código del método que evalua la condición de la interacción.

```
public boolean checkCondition(){
    ScreenPresenceDetection = frameworkSearcher.getComponent("NearScreenPresenceDetector");
          ScreenGradualLighting = frameworkSearcher.getComponent("ScreenGradualLighting");
    boolean returnValue;
    returnValue = ScreenPresenceDetection.presenceDetected() &&
          (! ScreenGradualLighting.isLighitng() );
          return returnValue;
    }
```

6. Modelado del sistema pervasivo mediante PervML

En esta sección se presenta la especificación del caso práctico en Perv-ML. En primer lugar se describe brevemente el sistema que se pretende especificar. A continuación se construyen y se detallan los modelos propuestos por Perv-ML para especificar este sistema.

6.1. Requisitos del sistema

El sistema pervasivo desarrollado proporciona servicios en el ámbito de un hogar. Se trata de una casa de dos plantas con jardín. En la primera planta se encuentra la zona de día (salón, cocina y jardín) y en la segunda planta se encuentran los dormitorios (para los hijos y para los padres) y el baño.

La funcionalidad que proporciona el sistema es muy variada e incluye gestión de la iluminación (de manera "inteligente" en algunos puntos de la casa), gestión de multimedia (que permite reproducir archivos multimedia en varias ubicaciones), gestión de la climatización (intentado no desperdiciar energía y aprovechando los recursos naturales) y seguridad (principalmente detección de intrusos y accidentes).

Detallamos los requisitos del sistema con una breve descripción de los distintos tipos de servicio que ofrece:

- **Iluminación:** proporciona y controla la iluminación de una ubicación. Existen servicios de iluminación general en el salón, el recibidor, el pasillo y el cuarto de baño.

- **Iluminación gradual**: además de la funcionalidad de un servicio de Iluminación básico, este servicio permite controlar la intensidad de la iluminación. Un servicio de este tipo estará ubicado en el jardín y otro en el salón.

- **Iluminación "inteligente"**: un servicio de iluminación que sólo se activa si la luminosidad de la sala es menor que un umbral fijado por el usuario. Necesita, por tanto, de servicios de detección de luminosidad. Estos servicios estarán instalados en los dormitorios.

- **Reproducción de multimedia:** permite reproducir contenidos multimedia, tanto audio como vídeo. También proporciona funcionalidad para controlar la reproducción (avanzar rápido, rebobinar, pausar la reproducción, etc.). Este se servicio se proporcionará en el salón y en el dormitorio de los hijos.

- **Servidor multimedia**: almacena contenido multimedia de todo tipo. Existirá un servicio de este tipo en el sistema.

- **Grabación de vídeo**: graba en un archivo un vídeo con imágenes que captura. El sistema dispondrá un servicio para grabar las imágenes del jardín.

- **Envío de Mensajes**: permite enviar un mensaje de texto a un destinatario con un determinado nivel de prioridad. Existirá un servicio de este tipo en el sistema.

- **Detección de presencia**: informa sobre la existencia de presencia en una ubicación determinada. Existirán varios servicios de este tipo: en el jardín, en el pasillo, en el baño, en ambos dormitorios y uno general para dentro de la casa.

- **Seguridad**: es un servicio compuesto que puede ser activado o desactivado por el usuario. Para poder activarse todas las puertas y ventanas han de estar cerradas y la casa vacía. El servicio se dispara si detecta presencia en el interior de la casa o alguna puerta o ventana se abre. Cuando se dispara se avisará al propietario y a una central de alarmas, se activará un zumbador y se iniciará una grabación en vídeo hasta que se desactive.

- **Detección de apertura**: indica si una puerta o ventana se encuentra abierta o cerrada. Se utilizará en este servicio en la puerta principal, en la ventana del salón y en las de los dormitorios.

- **Detección de temperatura**: mide la temperatura de una estancia. En el sistema habrá un servicio de este tipo para cada una de las plantas de la casa y otro específico para el baño.

- **Climatización**: permite fijar la temperatura de una estancia entre un rango de confort. Se instalará un servicio de este tipo en el salón, en los dos dormitorios y en el cuarto de baño. Para que este servicio este activo las ventanas de la estancia deben estar cerradas.

- **Registro de Gastos**: almacena el gasto económico que supone el uso de ciertos servicios del sistema. Por ejemplo, la reproducción de ciertos archivos multimedia, el envío de mensajes urgentes o la notificación al servicio de alarma tienen un coste económico. Este servicio registra estos gastos en una aplicación de contabilidad doméstica externa al sistema.

- **Detección de viento**: indica la velocidad del viento. Existirá un servicio de este tipo en el jardín.

- **Seguridad ante tormentas**: se encarga de subir las ventanas si el viento supera un umbral de velocidad que puede ser fijado por el usuario. Además, si alguna ventana está abierta avisa al usuario.

- **Iluminación por presencia**. enciende un servicio de iluminación cuando detecta presencia en una ubicación. Este servicio puede activarse y desactivarse. Existirán servicios de este tipo en el pasillo, en el baño y en ambos dormitorios.

- **Detección de Luminosidad**: indica la luminosidad en una estancia. Se dispondrán de servicios de este tipo en el jardín y en los dormitorios.

- **Activación**: permite al usuario activar y desactivar otros servicios del sistema. En nuestro caso, existirán varios servicios de este tipo. Uno en el jardín

para controlar el servicio de seguridad. Uno en cada dormitorio para desactivar el servicio de iluminación por presencia (por las noches no es necesario). Otro en el salón para controlar manualmente la iluminación.

Además, en el sistema tendrán lugar las siguientes interacciones.

- Se especificarán interacciones (varios umbrales) para reducir la intensidad de la iluminación del jardín atendiendo a la luminosidad exterior.

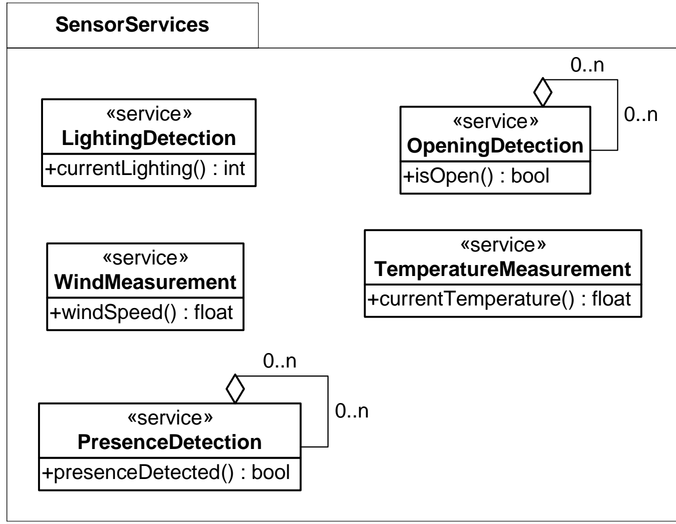
- Se atenuarán las luces graduales del salón y se bajarán las persianas cuando el servicio de reproducción esté reproduciendo un vídeo (no en caso de ser audio).

6.2. Modelado del sistema

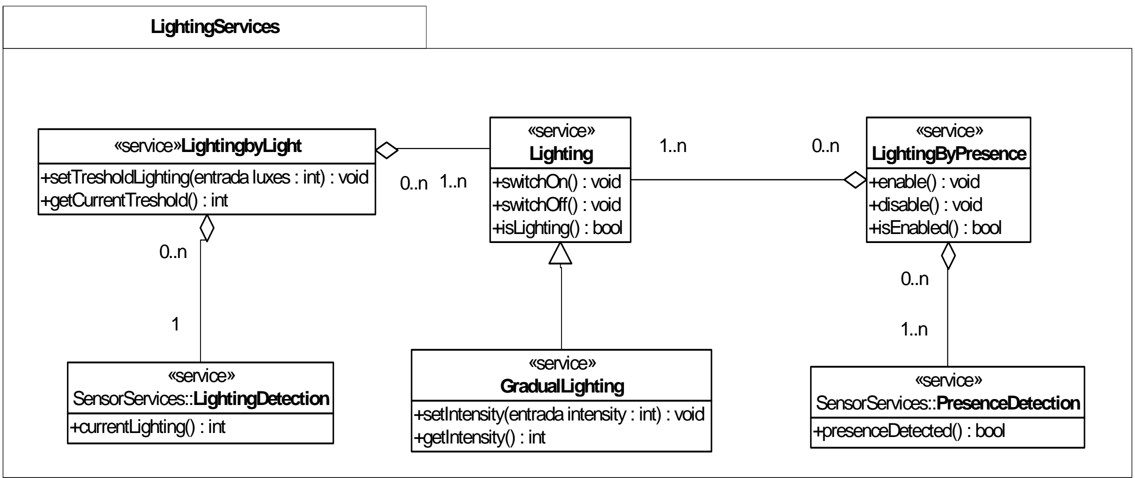
6.2.1. El Modelo de Servicios

En el Modelo de Servicios de PervML se describen los distintos tipos de servicios que puede haber en el sistema en desarrollo. Un servicio está compuesto de una serie de operaciones, cada una de ellas puede tener disparadores, pre y post condiciones. Además cada servicio tiene asociada una máquina de transición de estados para describir la secuencia válida de invocación de las operaciones.

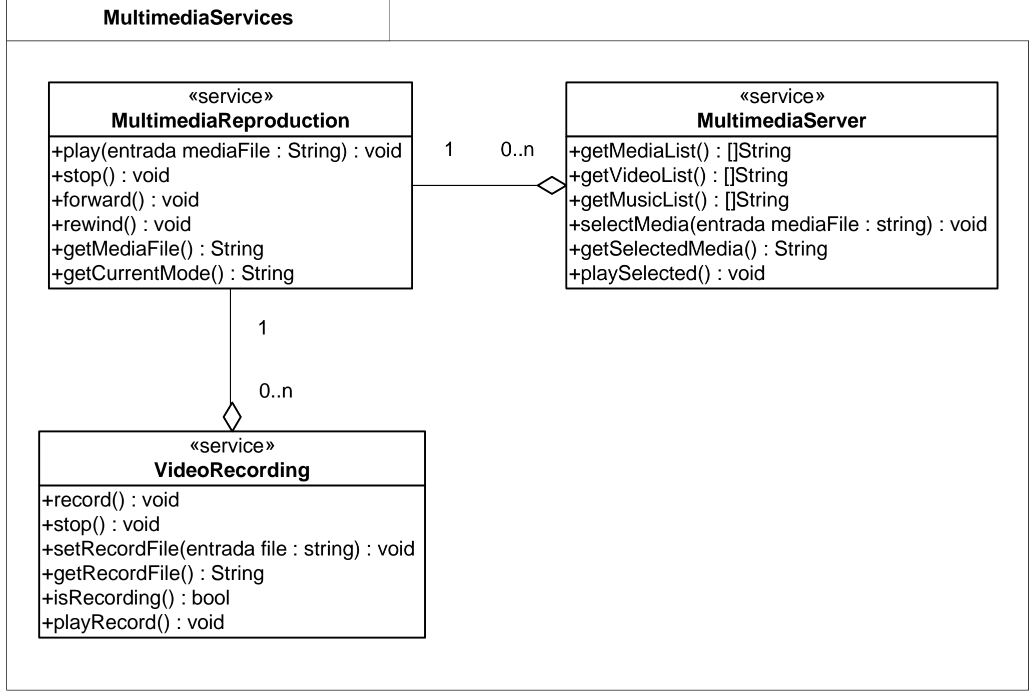
A continuación se muestra el Modelo de Servicios (representado mediante diagramas de clases UML) que muestra la organización general de los distintos tipos de servicios. Debido al gran número de tipos de servicio, esto se han organizado por categorías:



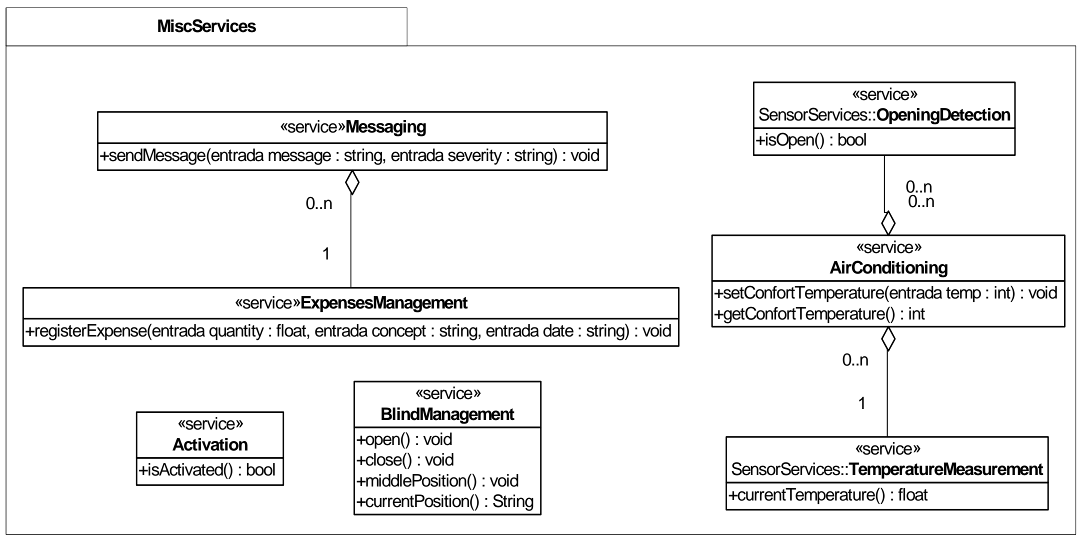
Los tipos de servicio dentro de la categoría *SensorServices*" se encargan de capturar información del entorno del sistema.



Los tipos de servicio dentro de la categoría "*LightingServices*" se encargan de gestionar la iluminación del entorno. Nótese que estos tipos de servicio hacen uso de algunos tipos de servicio de la categoria "*SensorServices*".

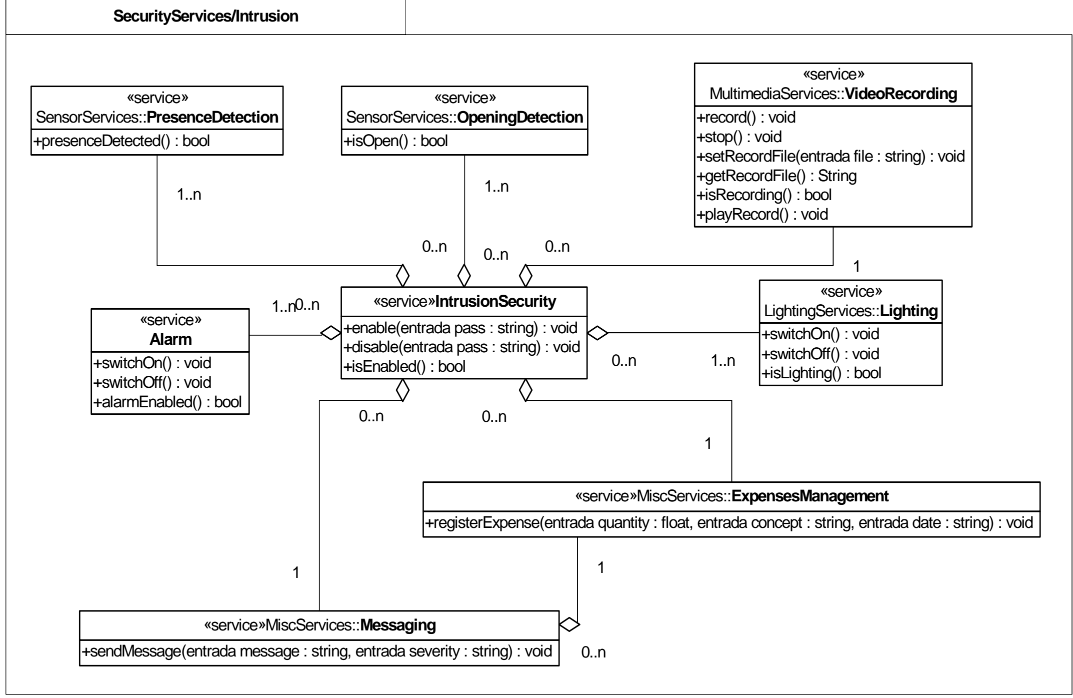


Los tipos de servicio dentro de la categoría "*MultimediaServices*" proporcionan funcionalidad relacionada con documentos multimedia; principalmente, video y audio.

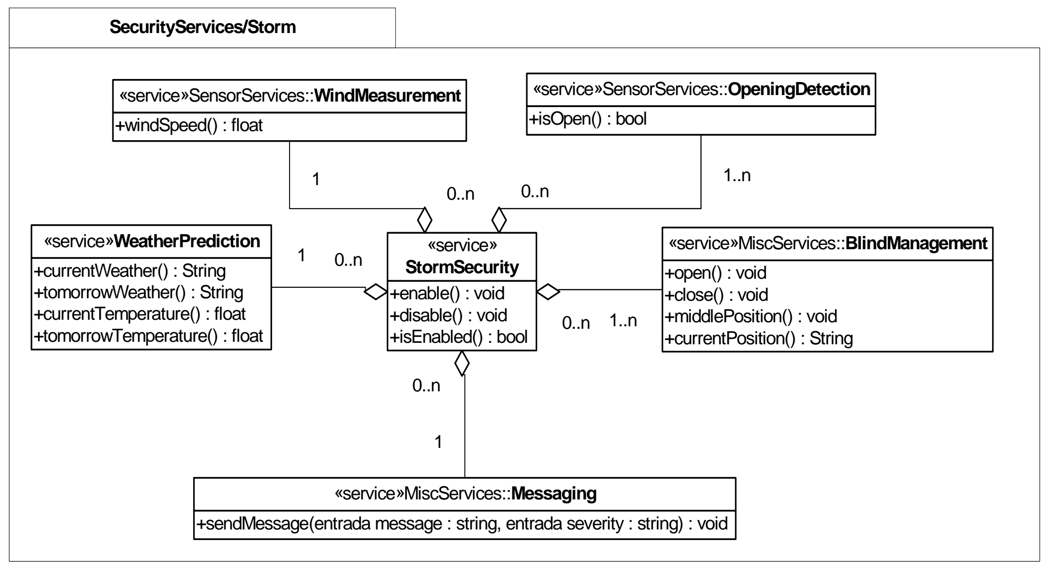


Los tipos de servicio dentro de la categoría "*MiscServices*" proporcionan funcionalidad que no ha podido ser catalogada en ninguno de los otros tipos.

Debido que los tipos de servicios que ofrecen funcionalidad de seguridad tienen muchas relaciones con otros servicios, se ha optado por dividirlos en dos subcategorías.



Los tipos de servicio dentro de la categoría "*SecurityServices/Intrusion*" proporcionan funcionalidad para gestionar la seguridad ante intrusiones en el casa.

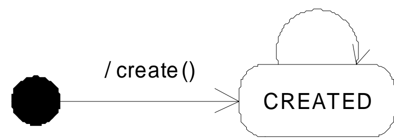


Los tipos de servicio dentro de la categoría "*SecurityServices/Storm*" proporcionan funcionalidad para gestionar la seguridad ante tormentas.

Describimos a continuación cada uno de los tipos de servicios que aparecen en el modelo de servicios. Para ello se explicará textualmente su funcionamiento, se mostrará su diagrama de transición de estados y se especificarán las pre y post condiciones de cada operación.

LightingDetection

La operación "currentLighting()" devuelve la cantidad de luminosidad en Luxes que detecta en el entorno.



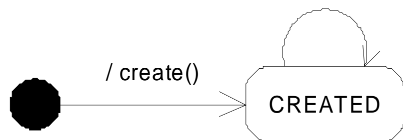
Operación	Pre	Post
currentLighting()	true	true

Fórmulas de los Estados

Estado	Fórmula
CREATED	true

WindMeasurement

La operación "windSpeed()" devuelve la velocidad del viento en kilómetros/hora.



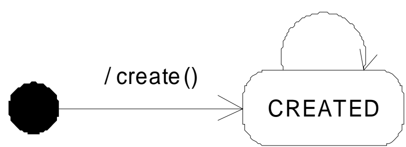
Operación	Pre	Post
windSpeed()	true	true

Fórmulas de los Estados

Estado	Fórmula
CREATED	true

PresenceDetection

La operación "presenceDetected()" indica si detecta presencia (true) o no (false).



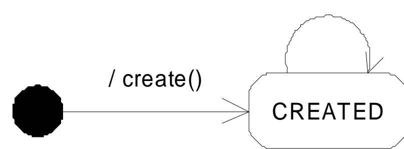
Operación	Pre	Post
presenceDetected()	true	true

Fórmulas de los Estados

Estado	Fórmula
CREATED	true

OpeningDetection

La operación "isOpen()" indica si detecta que la puerta o ventana se encuentra abierta (true) o no (false).



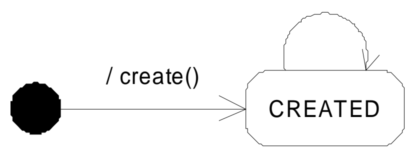
Operación	Pre	Post
isOpen()	true	true

Fórmulas de los Estados

Estado	Fórmula
CREATED	true

TemperatureMeasurement

La operación "currentTemperature" devuelve los grados centígrados que detecta en el ambiente.



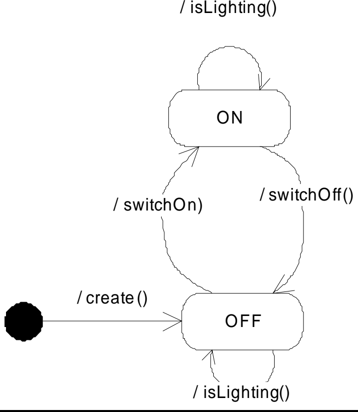
Operación	Pre	Post
currentTemperature()	true	true

Fórmulas de los Estados

Estado	Fórmula
CREATED	true

Lighting

Las operaciones "switchOn()" y "switchOff()" encienden y apagan respectivamente el servicio de iluminación. La operación "isLighting()" devuelve el estado actual del servicio (true si está encendido y false si está apagado).



Operación	Pre	Post
switchOn()	true	isLighting() = true

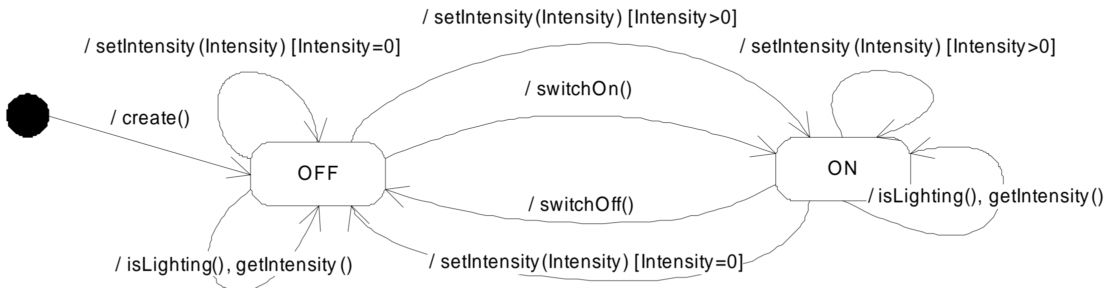
Operación	Pre	Post
switchOff()	true	isLighting() = false
isLighting()	true	true

Fórmulas de los Estados

Estado	Fórmula
OFF	isLighting() = false
ON	isLighting() = true

GradualLighting

Además de las operaciones de encendido y apagado heredadas de Lighting, este servicio permite fijar (y consultar) un porcentaje de intensidad lumínica. Tal y como se indica en las post-condiciones, encender el servicio utilizando “switchOn” equivale a fijar la intensidad al 100%



Operación	Pre	Post
switchOn()	true	(isLighting() = true) and (getIntensity() = 100)
switchOff()	true	(isLighting() = false) and (getIntensity() = 0)
isLighting()	true	true
setIntensity(intensity)	intensity >= 0	(getIntensity() = intensity) and (if intensity = 0 then isLighting() = false else isLighting() = true)
getIntensity()	true	true

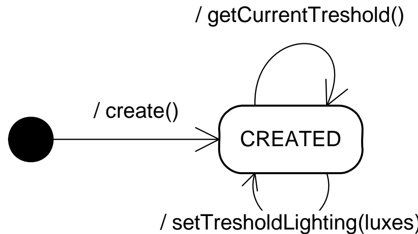
Fórmulas de los Estados

Estado	Fórmula
OFF	isLighting() = false

Estado	Fórmula
ON	isLighting() = true

LightingByLight

Los servicios de este tipo activan automáticamente los servicios de iluminación con los que se encuentran asociados, cuando la luminosidad desciende de un cierto límite que es fijado por el usuario. La operación "setTresholdLighting" permite fijar ese límite, mientras que la operación "getCurrentTreshold" devuelve el límite que está considerando actualmente el servicio.



Operación	Pre	Post
getCurrentTreshold()	true	true
setThresholdLighitng(luxes)	luxes > 0	getCurrentThreshold() = luxes

Fórmulas de los Estados

Estado	Fórmula
CREATED	true

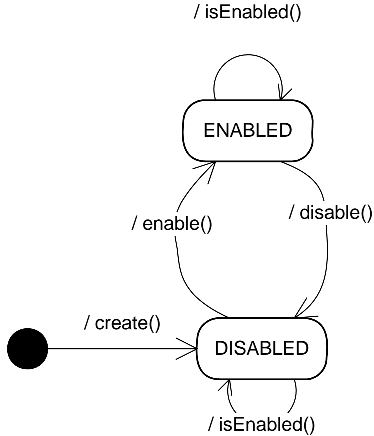
Triggers

when getCurrentTreshold() > LightingDetection.currentLighting() do

Lighting->forAll(l | l.switchOn())

LightingByPresence

Los servicios de este tipo activan automáticamente los servicios de iluminación con los que se encuentran asociados, cuando se detecta presencia en su entorno. Estos servicios proporcionan operaciones para activar y desactivar esta funcionalidad.



Operación	Pre	Post
enable()	true	isEnabled() = true
disable()	true	isEnabled() = false
isEnabled()	true	true

Fórmulas de los Estados

Estado	Fórmula
DISABLED	isEnabled() = false
ENABLED	isEnabled() = true

Triggers

when (isEnabled() = true) and (PresenceDetection->exists(p | p.presenceDetected() = true)) **do**

Lighting->forAll(l | l.switchOn())

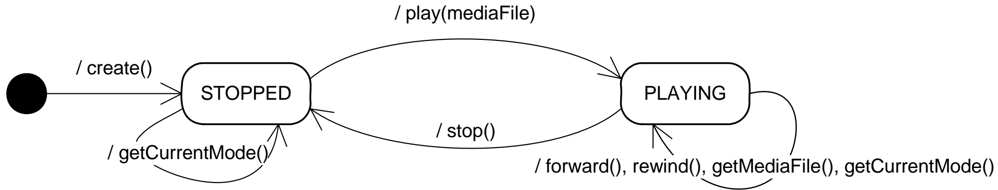
when (isEnabled() = false) and (PresenceDetection->forAll(p | p.presenceDetected() = false)) **do**

Lighting->forAll(l | l.switchOff())

MultimediaReproduction

Este tipo de servicio permite reproducir elementos multimedia. La operación "play(mediaFile)" inicia la reproducción de el archivo "mediaFile", que indica la ruta para acceder al archivo desde el servidor central (puede ser un archivo local, un archivo compartido o un archivo en Internet). Durante la reproducción de un archivo

multimedia es posible acelerar la reproducción ("forward()") o rebobinarla ("rewind()"). La operación "getCurrentMode()" devuelve la situación actual del servicio, que puede ser: apagado ("stopped"), reproduciendo ("playing"), avanzando ("forwarding") o rebobinando ("rewinding"). Mediante la operación "getMediaFile()" devuelve la ruta del archivo que se está reproduciendo.



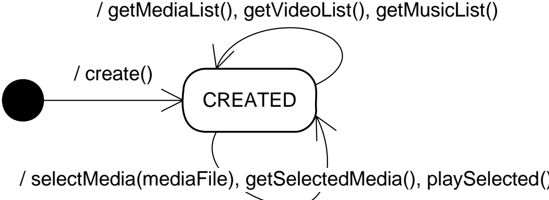
Operación	Pre	Post
play(mediaFile)	mediaFile <> ''	(getMediaFile() = mediaFile) and (getCurrentMode() = 'playing')
stop()	true	(getMediaFile() = '') and (getCurrentMode() = 'stopped')
forward()	true	getCurrentMode() = 'forwarding'
rewind()	true	getCurrentMode() = 'rewinding'
getMediaFile()	true	true
getCurrentMode()	true	true

Fórmulas de los Estados

Estado	Fórmula
STOPPED	getCurrentMode() = 'stopped'
PLAYING	getCurrentMode() <> 'stopped'

MultimediaServer

Este tipo de servicio proporciona acceso a un catálogo de archivos multimedia. La operación "getMediaList()" devuelve una lista con todos los archivos multimedia disponibles. Las operaciones "getVideoList()" y "getMusicList()" devuelven sólo los archivos multimedia de video y audio respectivamente. La operación "selectMedia(mediaFile)" pemite seleccionar uno de los archivos del servidor. Una vez seleccionado, la operación "playSelected()" inicia la reproducción en el servicio de reproducción de multimedia al que debe estar asociado el servidor. Finalmente, la operación "getSelectedMedia()" devuelve cual ha sido el último archivo multimedia seleccionado.



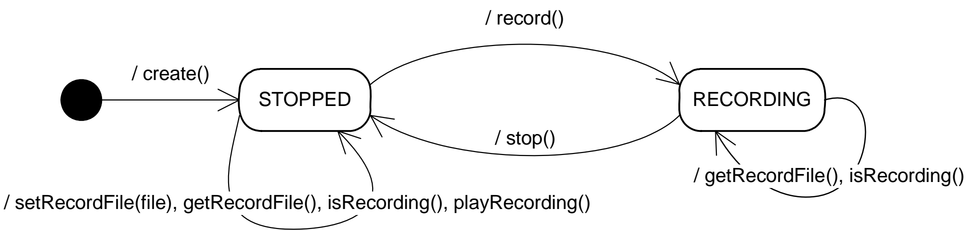
Operación	Pre	Post
selectMedia(mediaFile)	mediaFile <> ''	getSelectedMedia() = mediaFile
playSelected()	getSelectedMedia() <> ''	true
getSelectedMedia()	true	true
getMediaList()	true	true
getVideoFile()	true	true
getMusicFile()	true	true

Fórmulas de los Estados

Estado	Fórmula
CREATED	true

VideoRecording

Este tipo de servicio proporciona funcionalidad para grabar un archivo de video. La operación "record" inicia la grabación, mientras que la operación "stop" la detiene. El servicio graba el video en el archivo fijado mediante la operación "setRecord(file)".



Operación	Pre	Post
record()	getRecordFile() <> ''	isRecording() = true
stop()	true	isRecording() = false
setRecordFile(file)	file <> ''	getRecordFile() = file
isRecording()	true	true

Operación	Pre	Post
getRecordFile()	true	true
playRecord()	getRecordFile() <> ''	true

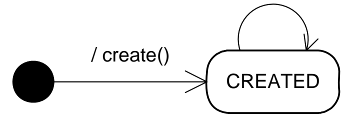
Fórmulas de los Estados

Estado	Fórmula
STOPPED	isRecording() = false
RECORDING	isRecording() = true

AirConditioning

Los servicios de este tipo controlan la climatización de un entorno. Para ello el usuario debe fijar una temperatura deseada utilizando la operación "setConfortTemperature(temp)".

/ setConfortTemperature(temp), getConfortTemperature()



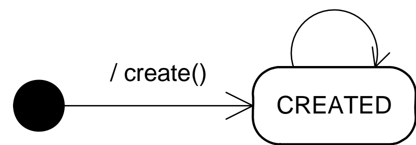
Operación	Pre	Post
setConfortTemperature(temp)	temp > 0	getConfortTemperature() = temp
getConfortTemperature()	true	true

Fórmulas de los Estados

Estado	Fórmula
CREATED	true

Activation

La operación "isActivated()" devuelve la situación actual del servicio, es decir, si el usuario lo ha activado (true) o desactivado (false).



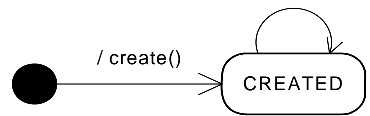
Operación	Pre	Post
isActivated()	true	true

Fórmulas de los Estados

Estado	Fórmula
CREATED	true

Messaging

Este tipo de servicio permite enviar mensajes con una cierta prioridad. El receptor de los mensajes dependerá de cada servicio en concreto, por lo que no se indica en la operación "sendMenssage". La cadena de texto que indica la gravedad del mensaje puede tener los valores "LOW", "MEDIUM", "HIGH". En ocasiones el envío de un mensaje puede tener un coste económico, por lo que este tipo de servicio se relaciona con servicios para gestionar gastos.



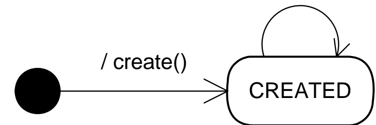
Operación	Pre	Post
sendMessage(message,severity)	(severity = 'LOW') or (severity = 'MEDIUM') or (severity = 'HIGH')	true

Fórmulas de los Estados

Estado	Fórmula
CREATED	true

ExpenssesManagement

Este tipo de servicio permite registrar gastos en una aplicación software externa.



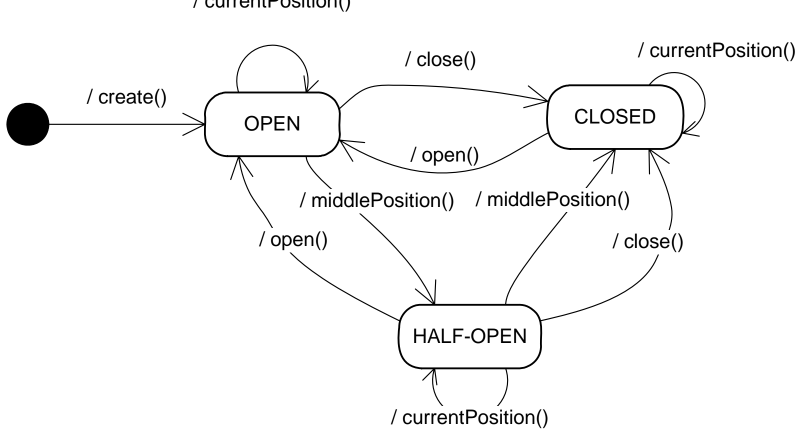
Operación	Pre	Post
recordExpense(quantity, concept, date)	quantity > 0	true

Fórmulas de los Estados

Estado	Fórmula
CREATED	true

BlindManagement

Este tipo de servicio proporciona operaciones para controlar persianas. La operación "*currentPosition*" devuelve una cadena con la situación actual de la ventana que puede ser "OPEN", "CLOSED" o "OTHER".



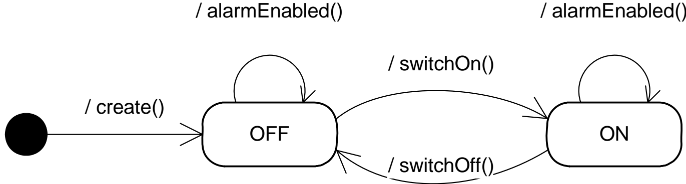
Operación	Pre	Post
open()	true	currentPosition() = 'OPEN'
close()	true	currentPosition() = 'CLOSED'
middlePosition()	true	currentPosition() = 'OTHER'
currentPosition()	true	true

Fórmulas de los Estados

Estado	Fórmula
OPEN	currentPosition() = 'OPEN'
CLOSED	currentPosition() = 'CLOSED'
HALF-OPEN	currentPosition() = 'OTHER'

Alarm

Este tipo de servicio proporciona funcionalidad para activar y desactivar un servicio de alarma, sea esta de cualquier tipo (sonora, visual, etc.).



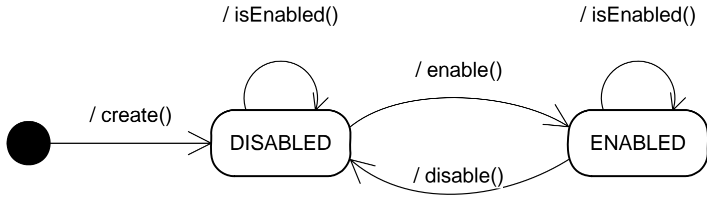
Operación	Pre	Post
switchOn()	true	alarmEnabled() = true
switchOf()	true	alarmEnabled() = false
alarmEnabled()	true	true

Fórmulas de los Estados

Estado	Fórmula
OFF	alarmEnabled() = false
ON	alarmEnabled() = true

IntrusionSecurity

El servicio se dispara si detecta presencia en el interior de la casa o alguna puerta o ventana se abre. Cuando se dispara se avisará al propietario y a una central de alarmas, se activará una alarma, se simulará presencia encendiéndose las luces y se iniciará una grabación en vídeo hasta que se desactive. Para poder activarse todas las puertas y ventanas han de estar cerradas y la casa vacía.



Operación	Pre	Post
enable()	PresenceDetection->forAll(p | p.presenceDetected() = false) and OpeningDetection->forAll(o | o.isOpen() = false)	isEnabled() = true
disable()	true	isEnabled() = false
isEnabled()	true	true

Fórmulas de los Estados

Estado	Fórmula
DISABLED	isEnabled() = true
ENABLED	isEnabled() = false

Triggers

when isEnabled() and (PresenceDetection->exists(p | p.presenceDetected()) or OpeningDetection->exists(o | o.isOpen()) **do**

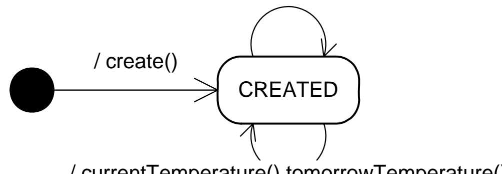
Alarma->forAll(a | a.switchOn()) and Lighting->forAll(l | l.switchOn()) and VideoRecording.record() and Messaging.sendMessage("Intrusion detected. Alarms switched on", "HIGH")

when isEnabled() == false or (PresenceDetection->forAll(p | p.presenceDetected() = false) and OpeningDetection->forAll(o | o.isOpen() = false) **do**

Alarma->forAll(a | a.switchOff()) and Lighting->forAll(l | l.switchOff()) and VideoRecording.stop() and Messaging.sendMessage("Alarms switched off", "MEDIUM")

WeatherPrediction

Este servicio proporciona funcionalidad relacionada con el tiempo metereológico. La operación "*currentTemperature*" indica la temperatura actual, mientras que "*tomorrowTemperature*" indica la máxima temperatura que previsiblemente hará el día siguiente. La operación "*currentWeather*" indica el estado actual del tiempo. Es una cadena que puede tomar uno de los siguientes valores: "*sunny*", "*rainny*", "*stormy*", "*cloudy*" o "*unknown*".



Operación	Pre	Post
currentTemperature()	true	true
tomorrowTemperature()	true	true

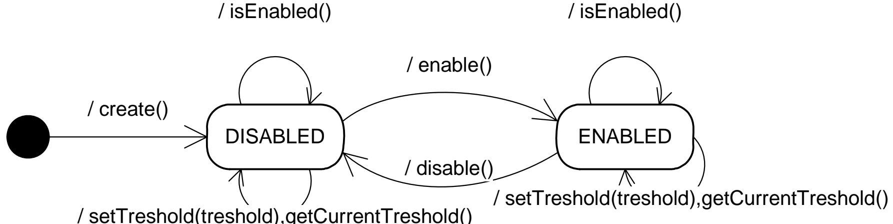
Operación	Pre	Post
currentWeather()	true	true
tomorrowWeather()	true	true

Fórmulas de los Estados

Estado	Fórmula
CREATED	true

StormSecurity

Los servicios de este tipo se encargan de bajamos las persianas si el viento supera un umbral de velocidad que puede ser fijado por el usuario. Además, si alguna ventana está abierta avisa al usuario.



Operación	Pre	Post
enable()	getCurrentThreshold() > 15	isEnabled() = true
disable()	true	isEnabled() = false
isEnabled()	true	true
setThreshold(treshold)	treshold >= 0	getCurrentThreshold() = treshold
getCurrentThreshold()	true	true

Fórmulas de los Estados

Estado	Fórmula
DISABLED	isEnabled() = true
ENABLED	isEnabled() = false

Triggers

when isEnabled() and WindMeasurement.windSpeed() > getCurrentTreshold() and currentWeather() = “STORMY” and BlindManagement->exists(b | b.currentPosition() != 'CLOSED') **do**

BlindManagement->forAll(b | b.close())

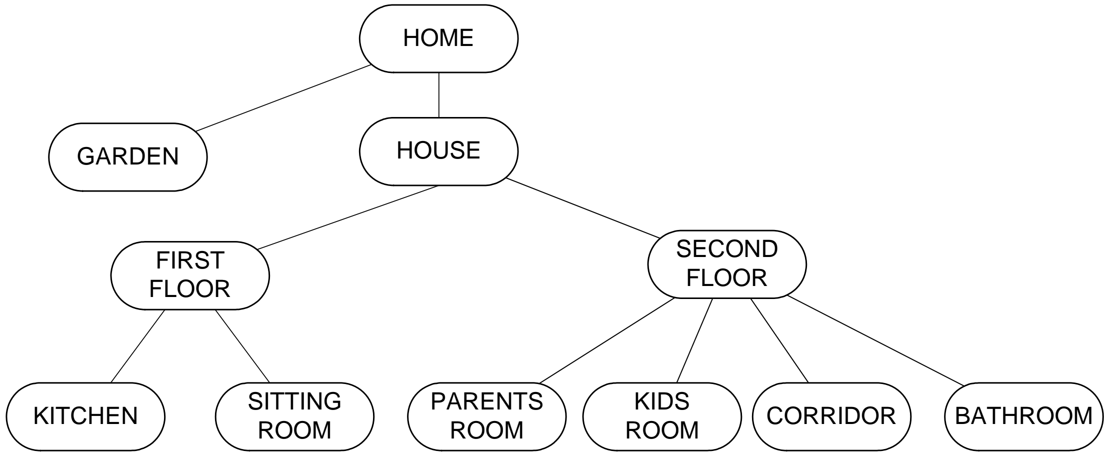
when isEnabled() and WindMeasurement.windSpeed() > getCurrentTreshold() and currentWeather() = “STORMY” and OpeningDetection->exists(o | o.isOpen()) **do**

Messaging.sendMessage(“We have an storm and there are open windows in the house”,”HIGH”)

6.2.2. El Modelo Estructural (Los Servicios del Sistema)

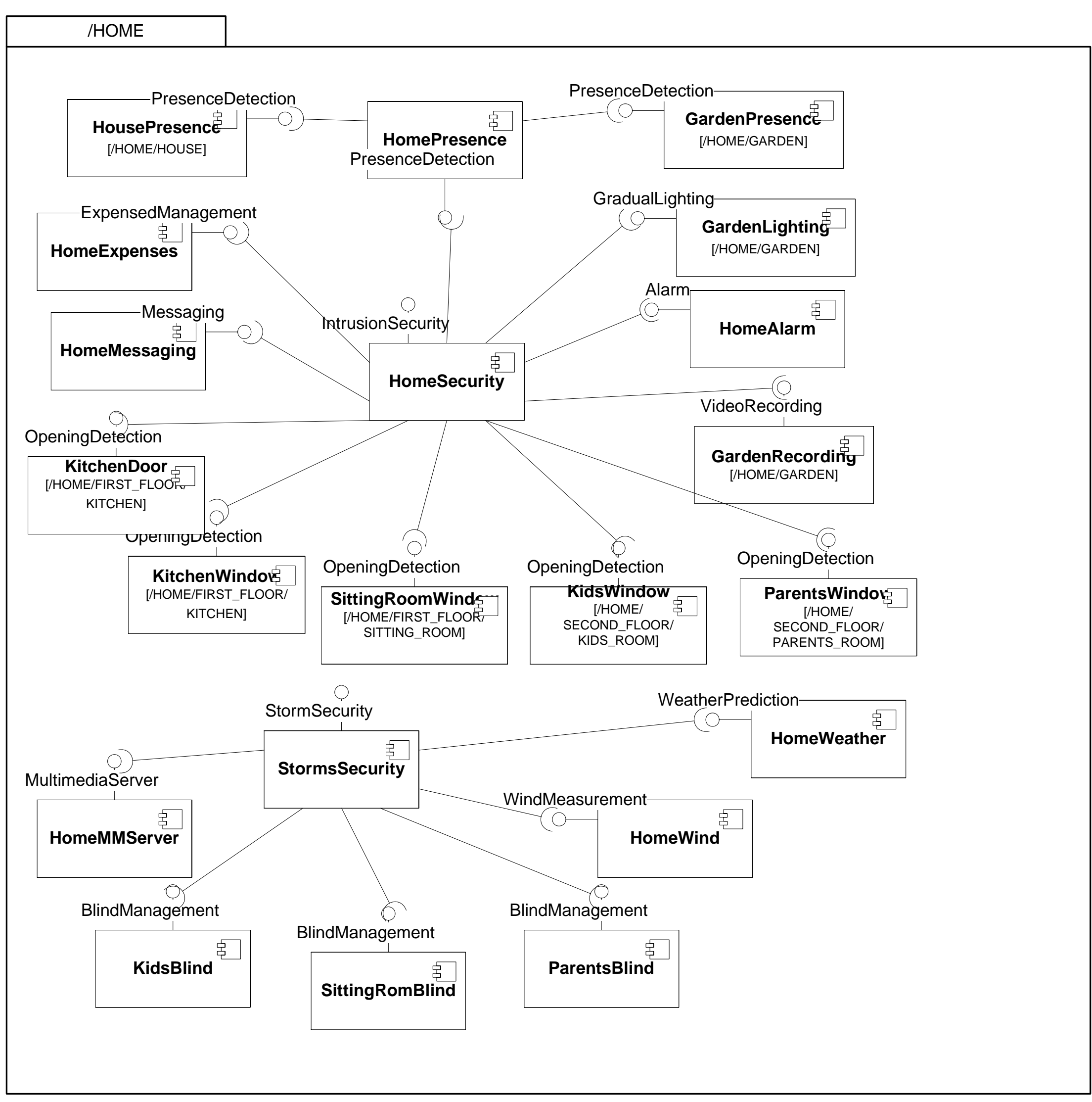
En el Modelo Estructural de PervML se especifican los distintos servicios que existirán en el sistema. Cada servicio está representado por un componente de UML 2.0. El servicio que proporciona se representa mediante una interfaz. Se especifican los servicios de cada ubicación del sistema en paquetes distintos.

El modelo de ubicación que proporciona PervML es simbólico, donde las distintas ubicaciones están jerarquizadas en un árbol. A continuación se muestra el árbol de ubicaciones de este caso de estudio.

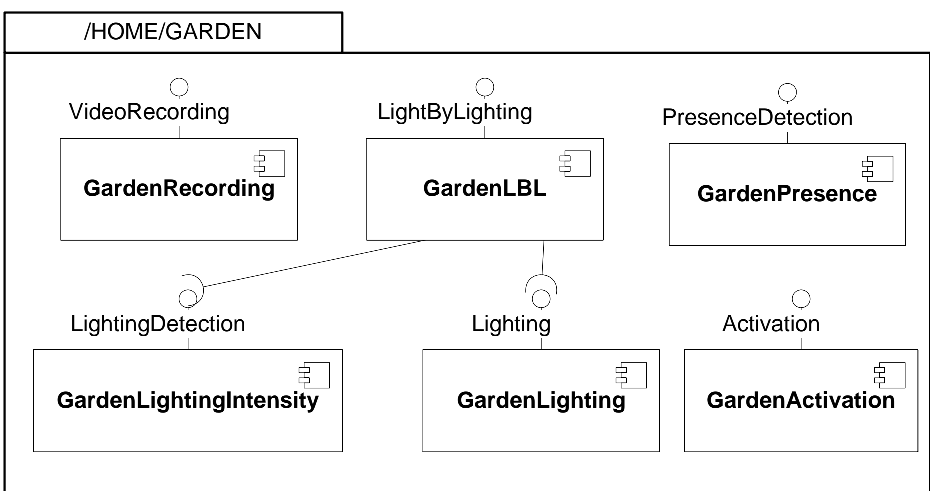


Para facilitar la comprensión del modelo estructural, los distintos componentes del sistema se organizan por ubicaciones. De manera que existirá un diagrama por cada nodo del árbol de ubicaciones.

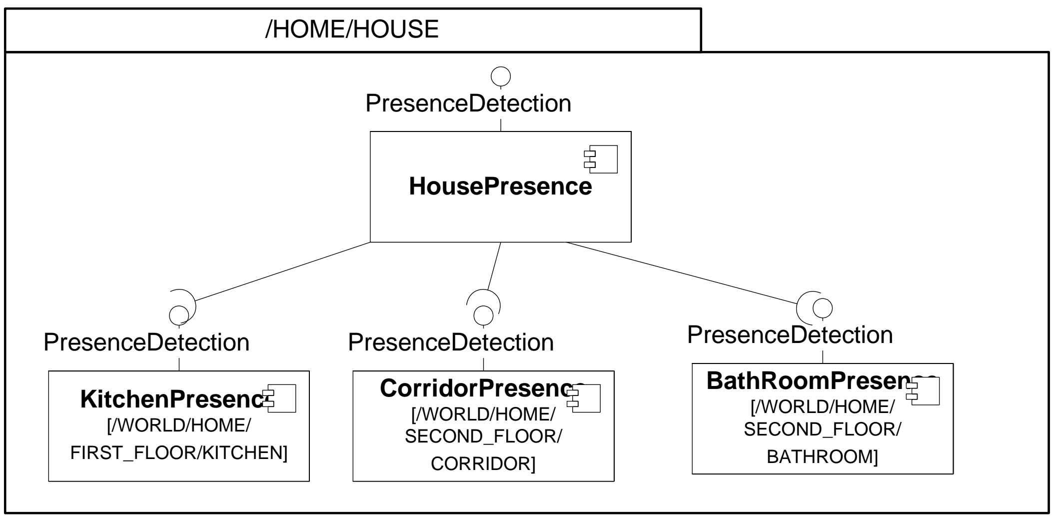
Home



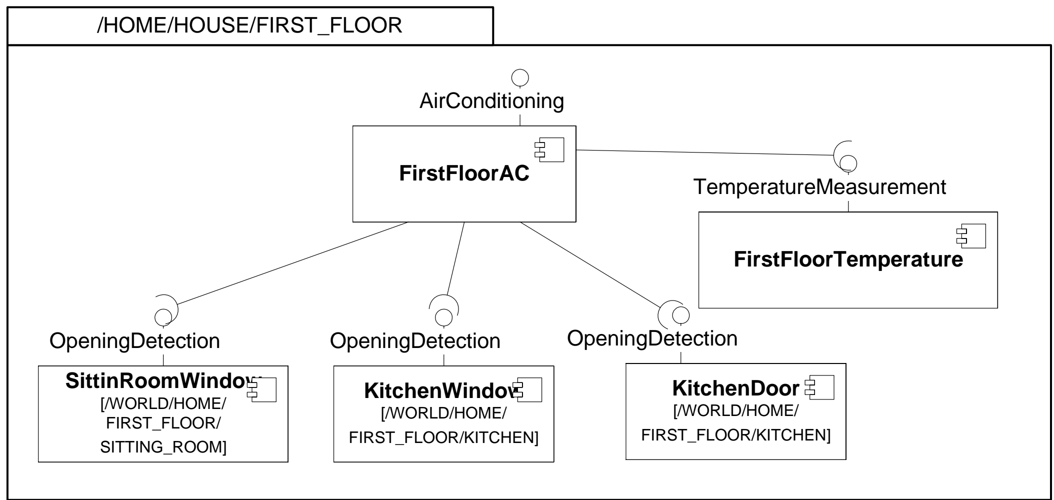
Garden



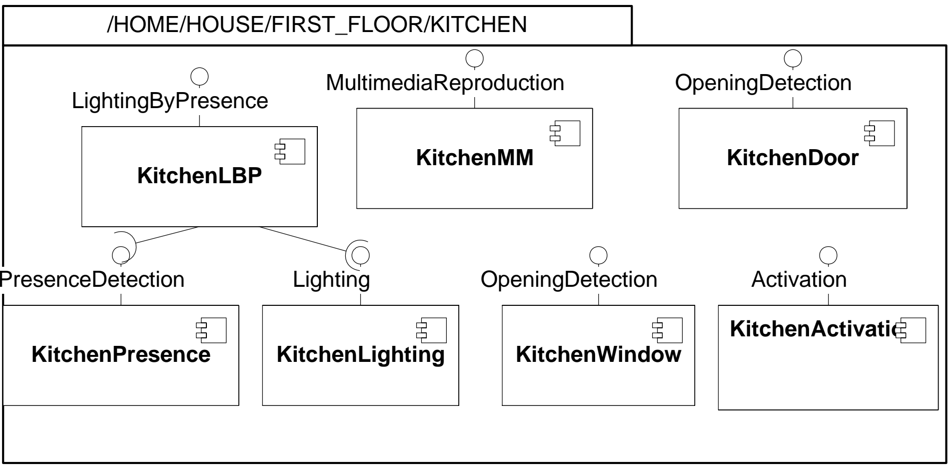
House



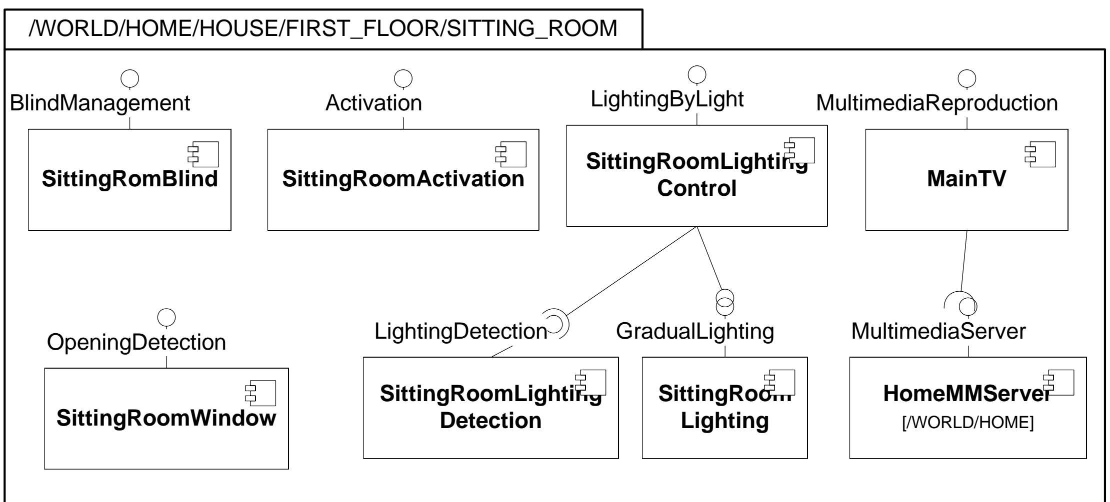
First Floor



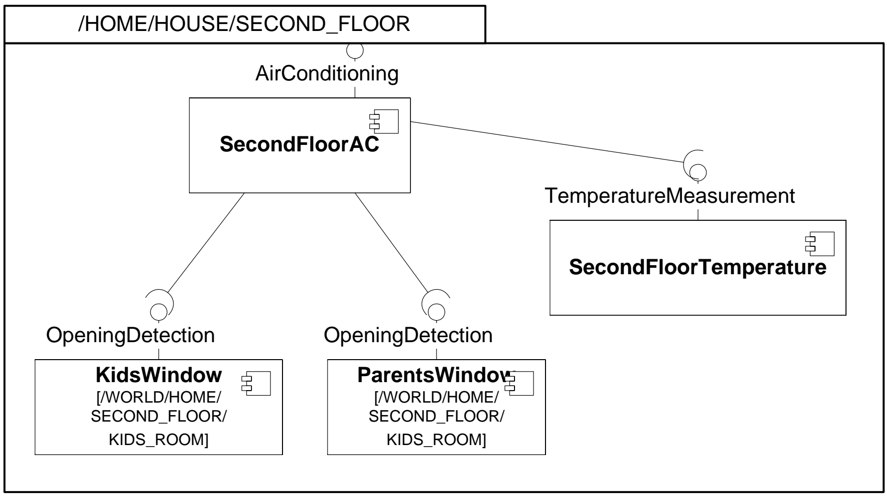
Kitchen



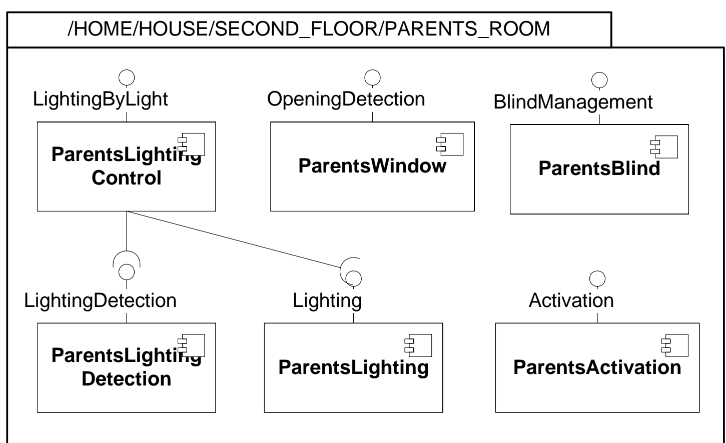
Sitting Room



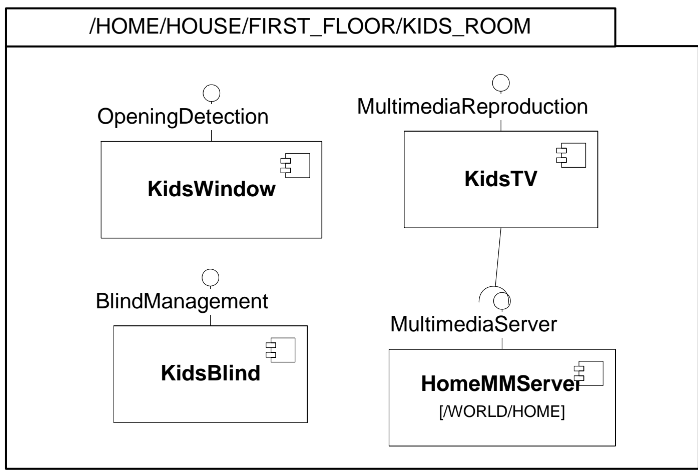
Second Floor



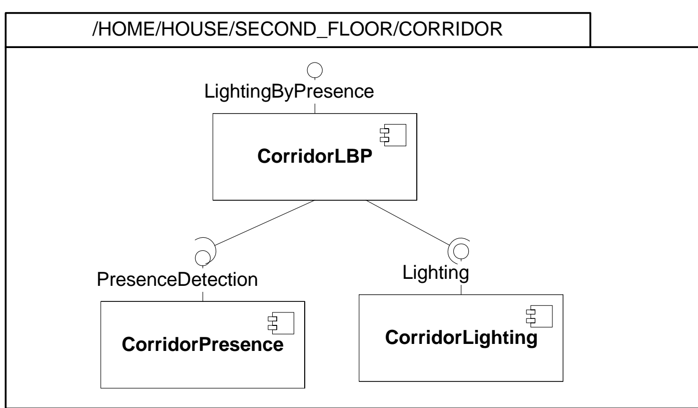
Parents Room



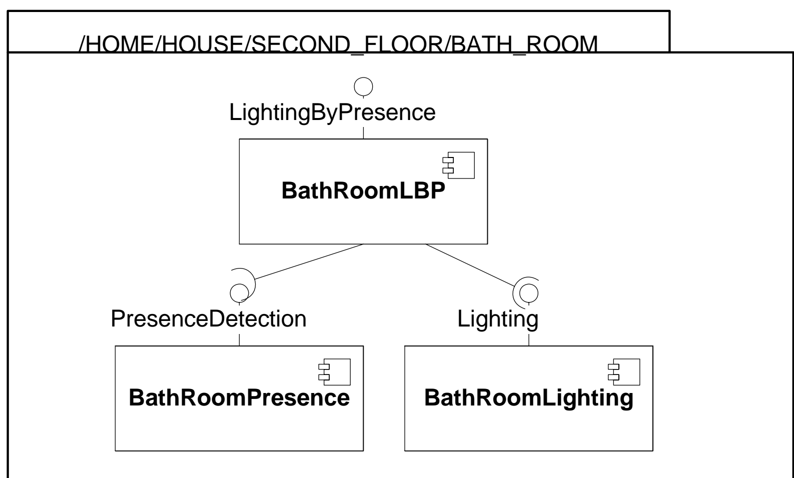
Kids Room



Corridor



Bathroom



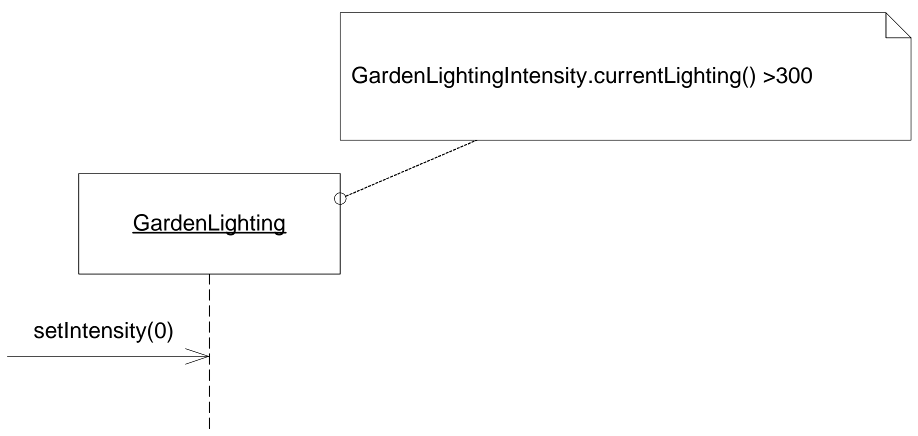
6.2.3. El Modelo de Interacción

En el Modelo de Interacción de PervML se especifica la funcionalidad del sistema resultante de combinar varios servicios y que se realizará a partir de ciertas situaciones o condiciones del sistema y no a partir de una invocación del usuario. El Modelo de Interacción está compuesto por interacciones, en la que se describe la condición que inicia la interacción y las acciones que se realizarán cuando la interacción tenga lugar.

Además, en el sistema tendrán lugar las siguientes interacciones.

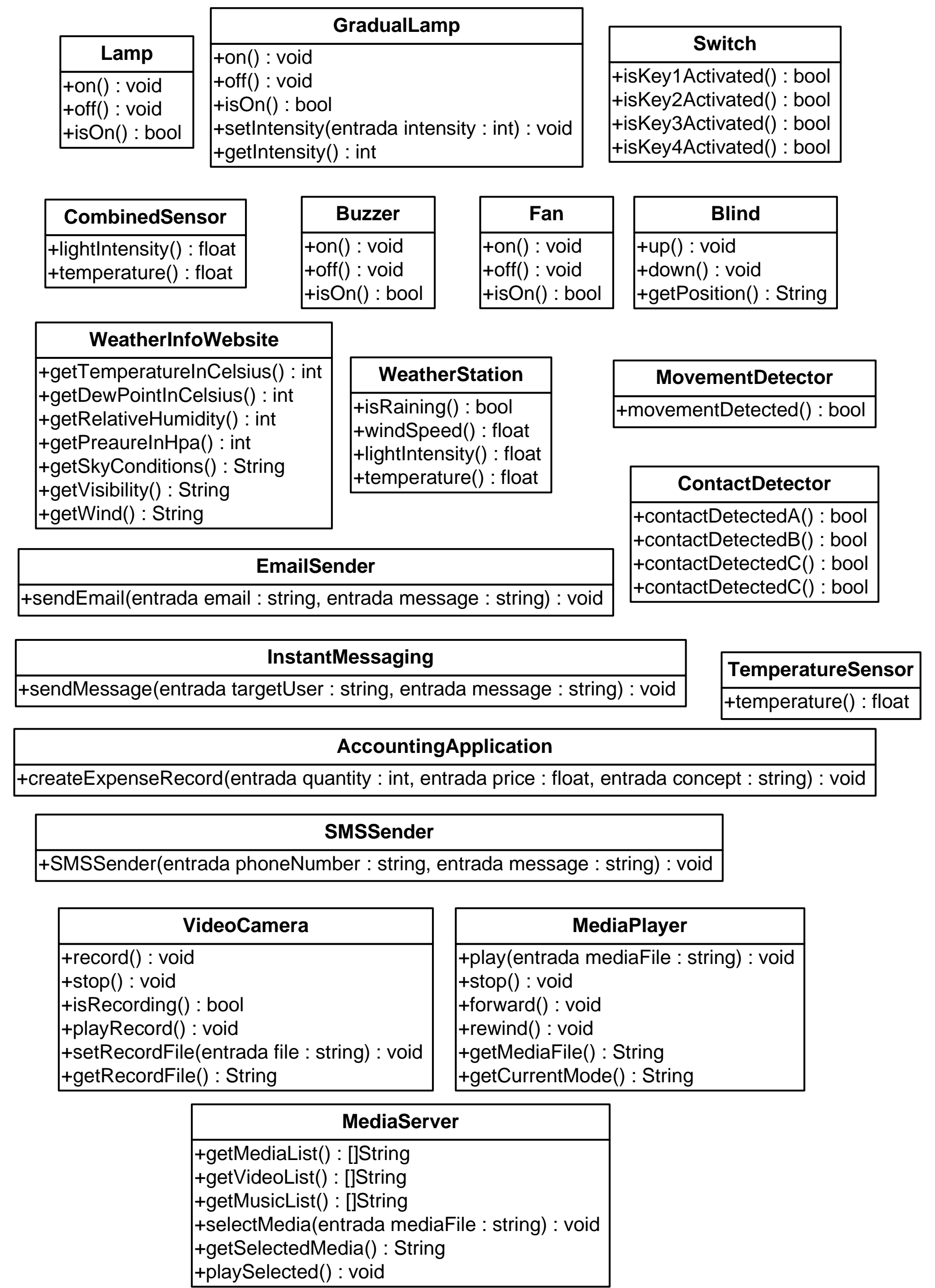
- Se especificarán interacciones (varios umbrales) para reducir la intensidad de la iluminación del jardín atendiendo a la luminosidad exterior.

- Se atenuarán las luces graduales del salón y se bajarán las persianas cuando el servicio de reproducción esté reproduciendo un vídeo (no en caso de ser audio).



6.2.4. El Modelo de Proveedores de Enlace

Con los anteriores modelos se han especificado los requisitos del sistema utilizando primitivas de alto nivel de abstracción, como servicio e interacción. Para poder implementar la funcionalidad requerida es necesario utilizar ciertos dispositivos o sistemas software externos. En el Modelo de Proveedores de Enlace se describen las interfaces de estos elementos que permitirán al sistema comunicarse con su entorno.



A continuación se describe cada uno de los tipos de dispositivos que se utilizarán para implementar la funcionalidad del sistema, indicando además los métodos que tienen y qué realiza cada uno de esos métodos.

Lamp (Lámpara o Bombilla)

Se trata de una única bombilla o de una lámpara (quizá con múltiples fuentes de luz pero que funcionan como un todo).

- **on()**: enciende la lámpara.
- **off()**: apaga la lámpara.
- **isOn()**: Si la lámpara está encendida devuelve true y si está apagada devuelve false.

GradualLamp (Lámpara gradual)

Se trata de una lámpara de intensidad graduable.

- **on()**: enciende la lámpara a la máxima intensidad.
- **off()**: apaga la lámpara.
- **isOn()**: Si la lámpara está apagada devuelve false, en caso contrario devuelve true.
- **setIntensity(intensity:int):** permite fijar la intensidad de la lámpara. El valor de entrada es un entero entre 0 y 100.
- **getIntensity():**

Switch (Interruptor)

Se trata de un interruptor con cuatro teclas independientes que puede estar encendidas o apagadas.

- **isKey1Activated()**: Si la tecla 1 del interruptor está pulsada devuelve true, si no lo está devuelve false.
- **isKey2Activated()**: Si la tecla 2 del interruptor está pulsada devuelve true, si no lo está devuelve false.
- **isKey3Activated()**: Si la tecla 3 del interruptor está pulsada devuelve true, si no lo está devuelve false.
- **isKey4Activated()**: Si la tecla 4 del interruptor está pulsada devuelve true, si no lo está devuelve false.

MovementDetector (Detector de movimiento)

Se trata de un dispositivo que detecta presencia en una zona determinada.

- **movementDetected()**: Si el sensor detecta movimiento devuelve true, si no lo detecta devuelve false.

ContactDetector (Detector de contacto)

Se trata de un dispositivo que consta de cuatro detectores de contacto. Cada uno de ellos detecta cuando dos superficies entran en contacto.

- **contactDetectedA()**: Si el sensor A detecta contacto devuelve true, si no lo detecta devuelve false.
- **contactDetectedB()**: Si el sensor B detecta contacto devuelve true, si no lo detecta devuelve false.
- **contactDetectedC()**: Si el sensor C detecta contacto devuelve true, si no lo detecta devuelve false.
- **contactDetectedD()**: Si el sensor D detecta contacto devuelve true, si no lo detecta devuelve false.

Buzzer (Zumbador)

Se trata de un dispositivo que emite un zumbido

- **on()**: enciende el zumbador.
- **off()**: apaga detiene el zumbador.
- **isOn()**: Si el zumbador está encendido devuelve true y si está apagado devuelve false.

WeatherStation (Estación metereológica)

Se trata de un dispositivo que captura información sobre distintas variables atmosféricas

- **isRaining()**: Si detecta llueva devuelve true y si no la detecta devuelve false.
- **windSpeed()**: Devuelve la velocidad del viento en metros por segundo.
- **lighIntensity()**: Devuelve la intensidad de la luz en luxes.
- **temperature()**: Devuelve la temperatura en grados centígrados.

CombinedSensor (Sensor Combinado)

Se trata de un dispositivo que combina dos sensores para medir luminosidad y temperatura.

- **lighIntensity()**: Devuelve la intensidad de la luz en luxes.
- **temperature()**: Devuelve la temperatura en grados centígrados.

Fan (Ventilador)

Se trata de un ventilador

- **on()**: enciende el ventilador.
- **off()**: apaga detiene el ventilador.
- **isOn()**: Si el ventilador está encendido devuelve true y si está apagado devuelve false.

MediaPlayer (Reproductor de Medios)

Se trata de un dispositivo que permite reproducir cualquier tipo de fichero multimedia.

- **Play(mediaFile)**: pone en marcha la reproducción del fichero multimedia que se le indica como parámetro.
- **stop()**: para la reproducción.
- **forward()**: incrementa la velocidad de reproducción.
- **rewind()**: rebobina el fichero multimedia.
- **getMediaFile()**: devuelve la ruta del fichero multimedia que se está reproduciendo.
- **getCurrentMode()**: devuelve el modo de funcionamiento ("STOPPED", "PLAYING","REWINDING", "FORWARDING").

MediaServer (Servidor de Medios)

Se trata de un dispositivo que permite el acceso a un catálogo de archivos multimedia.

- **getMediaList()**: devuelve una lista con todos los archivos multimedia disponibles.
- **getVideoList()**: devuelve sólo los archivos multimedia de video.
- **getMusicList()**: devuelve sólo los archivos multimedia de audio.

- **selectMedia(mediaFile):** pemite seleccionar uno de los archivos del servidor.

- **playSelected():** la inicia la reproducción del archivo seleccionado en el servicio de reproducción de multimedia al que debe estar asociado el servidor.

- **getSelectedMedia():** devuelve cual ha sido el último archivo multimedia seleccionado.

VideoCamera (Cámara de Video)

Se trata de un dispositivo que permite grabar un archivo de vídeo y reproducir la grabación.

- **stop()**: detiene la grabación o la reproducción.

- **record()**:inicia la grabación.

- **playRecord()**: inicia la grabación.

- **getMediaFile():** devuelve el nombre del fichero multimedia que se está reproduciendo.

- **getCurrentMode():** devuelve el modo de funcionamiento ("STOPPED", "PLAYING","REWINDING", "FORWARDING").

EMailSender (Enviador de e-mails)

Se trata de sistema software que envia mensajes correos electrónico.

- **sendEmail (String email, String message)**: Envía el texto message a la dirección de correo email.

InstantMessaging (Mensajería Instantánea)

Se trata de sistema software que envia mensajes a usuario de manera instantánea.

- **sendMessage (String targetUser, String message)**: Envía el texto message al destinatario identificado por targetUser. El identificador es dependiente de la tecnología de MI utilizada.

SMSSender (Enviador de SMS)

Se trata de sistema software que envia mensajes a móvil.

- **sendMessage (String phoneNumber, String message)**: Envía el texto message al número de teléfono phoneNumber.

AccountingApplication (Aplicación de Contabilidad)

Se trata de una aplicación de contabilidad que permite crear registros de gastos por aplicaciones externas.

- createExpenseRecord(int quantity, float price, String concept): Crea un registro de gatos.

Blind (Persiana)

Se trata de un dispositivo que controla las persianas. Considera que la persiana está completamente abierta cuando se ha subido 4 posiciones.

- **up()**: Sube la persiana en una posición.
- **down()**: Baja la persiana en una posición.
- **getposition()**: Devuelve la posición en la que se encuentra la persiana.

WeatherInfoWebSite (Sitio Web de Información Metereológica)

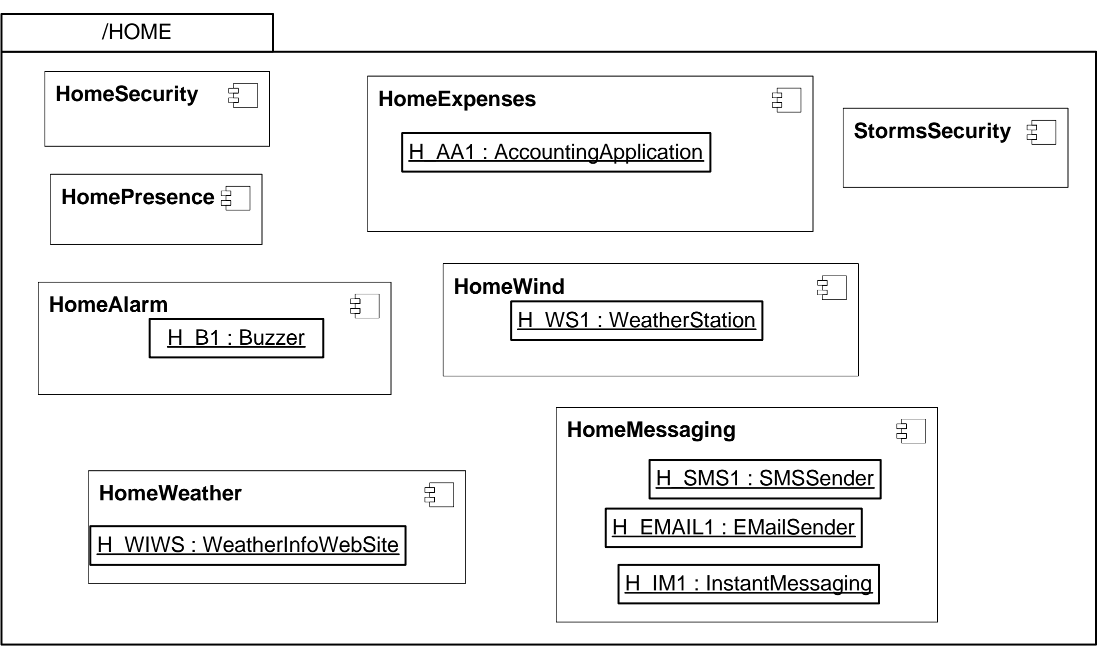
Se trata de un sistema software que accede a un sitio web que proporciona información metereológica.

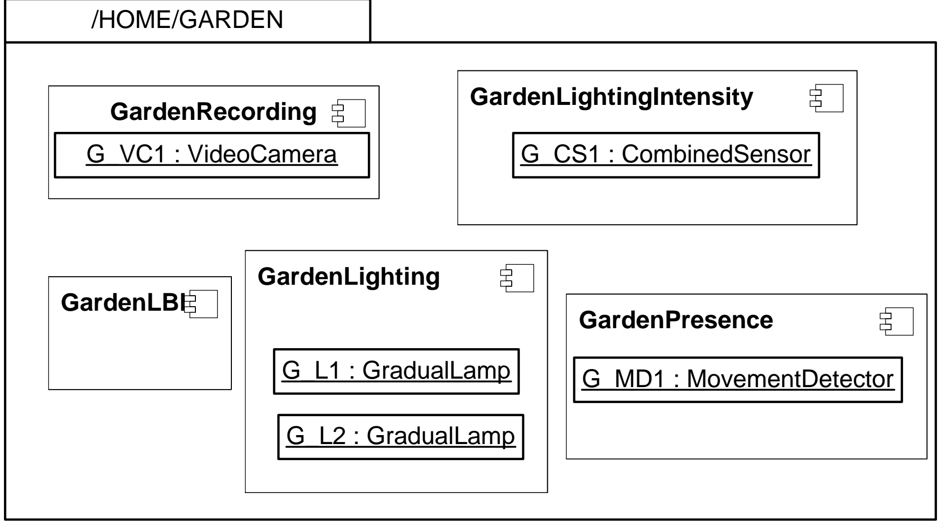
- **getTemperatureInCelsius ()**: Devuelve la temperatura en grados centígrados.
- **getDewPointInCelsius ()**: Devuelve el punto de condensación en grados centígrados.
- **getRelativeHumidity ()**: Devuelve el porcentaje de humedad.
- **getPressureInHpa()**: Devuelve la presión en pascales.
- **getSkyConditions()**: Devuelve la condición del cielo. Los posibles valores del texto de vuelve son: "CLEAR" o "CLOUDY".
- **getVisibility()**: Devuelve la visibilidad en kilómetros.
- **getWind()**: Devuelve la velocidad del viento en m/s.

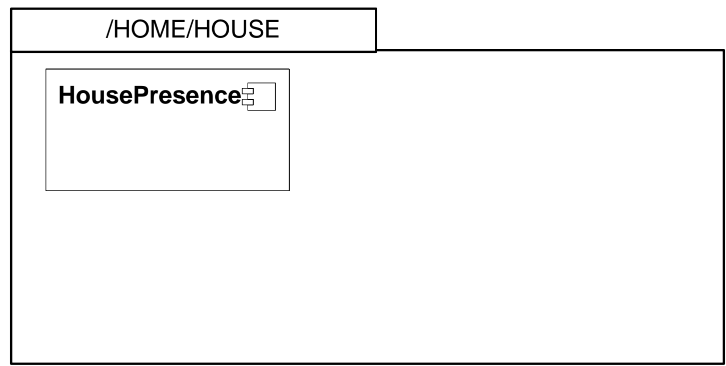
6.2.5. Especificación Estructural de los componentes

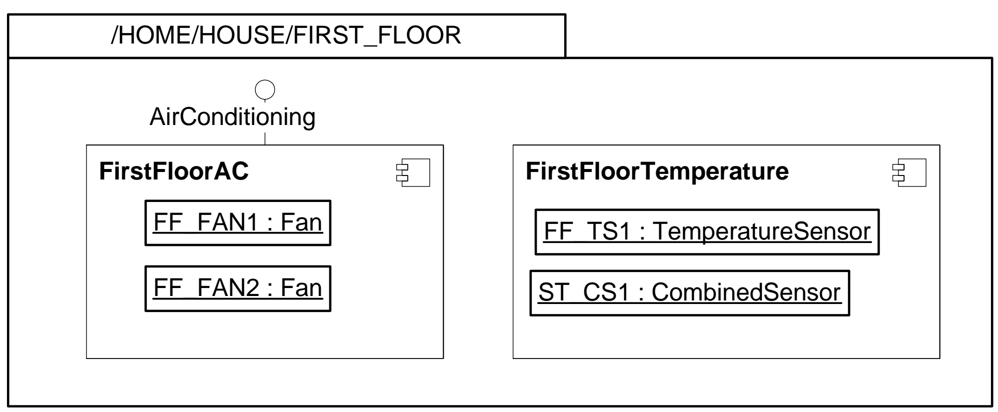
Mediante la Especificación Estructural de los componentes se indica qué proveedores de enlace van a implementar cada servicio del sistema. Se debe tener en cuenta que un mismo proveedor de enlace puede ser usado para proporcionar diversos servicios.

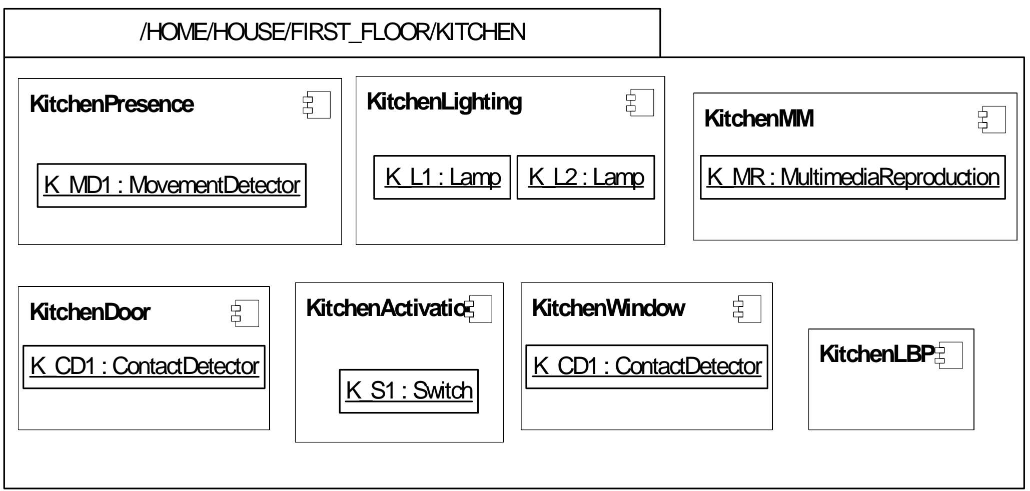
A continuación se muestra la especificación estructural en el que cada componente se representa mediante un componente de UML 2.0 y cada Binding Provider por un objeto de UML 2.0. Debido al gran número componentes que existen en el sistema, éstos se han organizado por localizaciones.

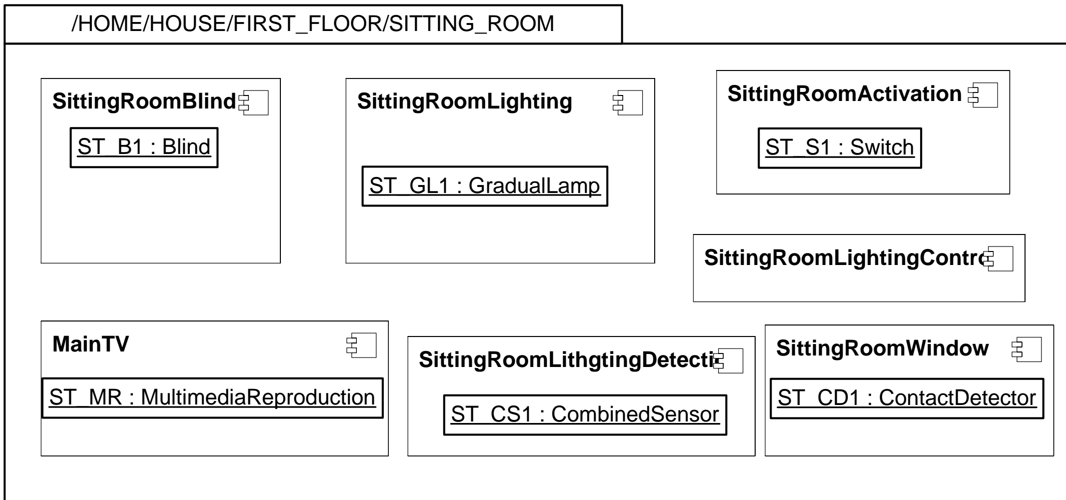




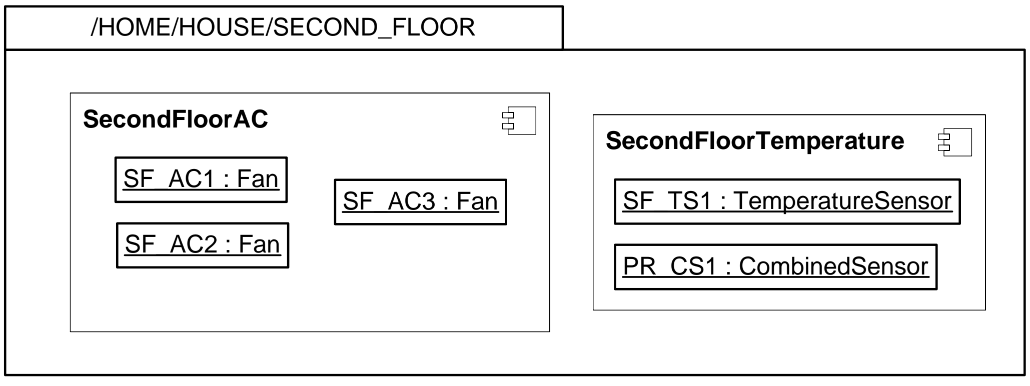

/HOME/HOUSE
HousePresence

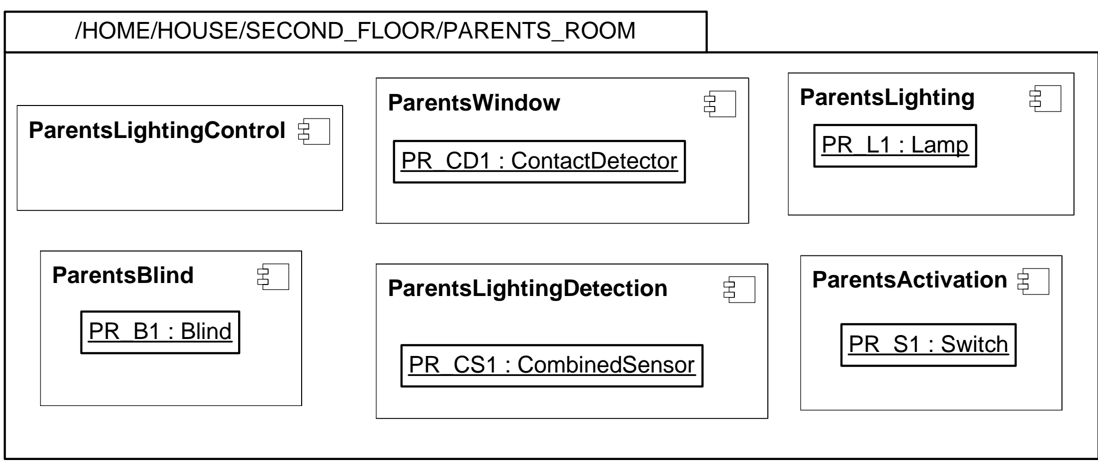

/HOME/HOUSE/FIRST_FLOOR
AirConditioning
FirstFloorAC
FF_FAN1 : Fan
FF_FAN2 : Fan
FirstFloorTemperature
FF_TS1 : TemperatureSensor
ST_CS1 : CombinedSensor


/HOME/HOUSE/FIRST_FLOOR/KITCHEN
KitchenPresence
K_MD1 : MovementDetector
KitchenLighting
K_L1 : Lamp
K_L2 : Lamp
KitchenMM
K_MR : MultimediaReproduction
KitchenDoor
K_CD1 : ContactDetector
KitchenActivatio
K_S1 : Switch
KitchenWindow
K_CD1 : ContactDetector
KitchenLBP

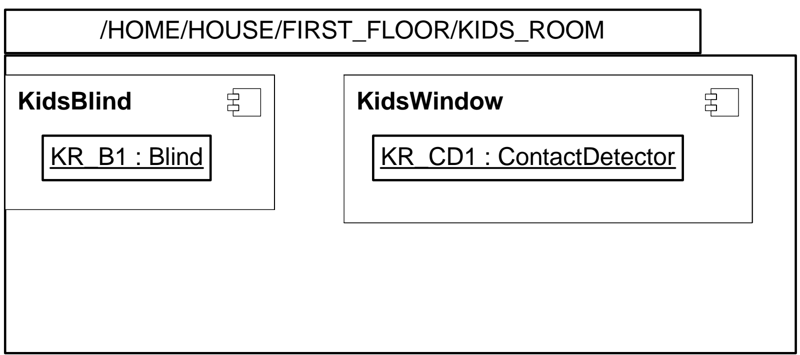


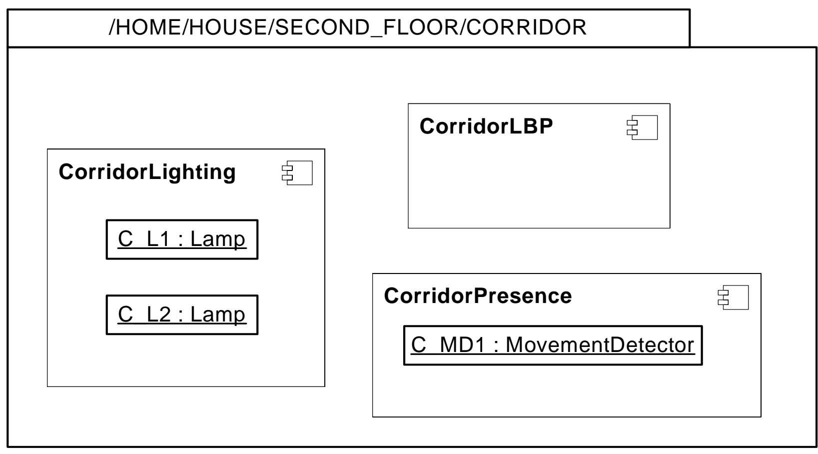
Tal y como se dijo anteriormente, podemos destacar que el Binding Provider ST_CS1 es usado por dos componentes: el componente FirstFloorTemperature y el componente SittingRoomLightingDetection.

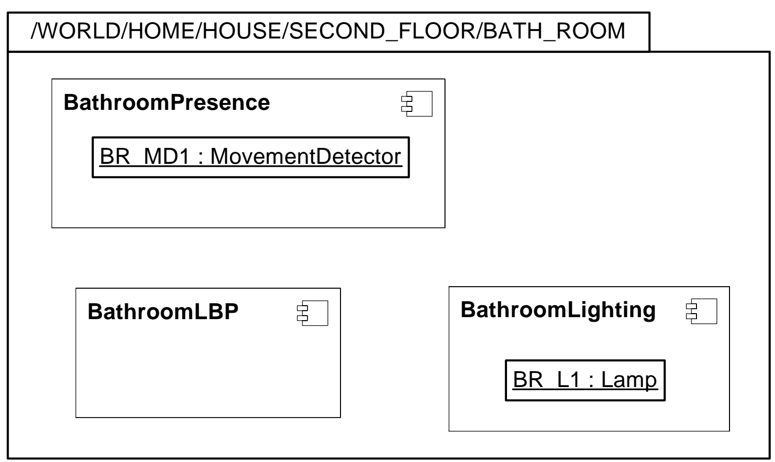




Al igual que el Binding Provider ST_CS1, el PR_CS1 también es usado por dos componentes: el componente SecondFloorTemperature y el componente ParentsRoomLightingDetection.







6.2.6. Especificación Funcional

En la Especificación Funcional de componentes de indican las acciones que se llevarán a cabo cuando se invoque una operación de un componente. Estas acciones podrán invocar operaciones de sus proveedores de enlace o de otros componentes con los que se encuentren relacionados.

Para la especificación funcional de componentes se utilizan las Acciones de UML 2.0. Debido a que estas acciones no tienen una sintaxis concreta, actualmente se utiliza la propuesta por Kennedy Carter.

Por comodidad, se utilizan las siguientes notaciones:

- "_NOMBRE_" para representar "this->BBProviders where nombre = 'NOMBRE'"
- "^NOMBRE^" para representar "this->Components where name = 'NOMBRE'"

Además, se considera que aquellas operaciones que devuelven un valor tienen definida una variable llamada "returnValue". Finalmente, aquellas variables que deben conservar su valor entre invocaciones de las operaciones se precederán del símbolo "@"; por ejemplo "@file".

Es importante destacar que, en ocasiones, la funcionalidad implementada en un componente no es, o no puede ser, la descrita utilizando lenguaje natural debido al modo de funcionamiento de los proveedores de enlace utilizados. Pese a eso, siempre se cumplen las pre y postcondiciones especificadas en el Modelo de Servicios.

HomeSecurity

void enable()

```
@enabled = TRUE;
```

void disable()

```
@enabled = FALSE;
```

bool isEnabled()

```
returnValue = @enabled ;
```

HomePresence

bool presenceDetected()

```
returnValue = PresenceDetection->exists( p | p.presenceDetected() = true )
```

HomeExpenses

void registerExpense(quantity:float, concept:String, date: String)

```
_H_AA1_.createExpenseRecord(1,quantity,concept+" : "+date)
```

HomeAlarm

void switchOn()

```
_H_B1_.on()
```

void switchOff()

```
_H_B1_.off()
```

void alarmEnabled()

```
_H_B1_.isOn()
```

HomeWind

float windMeasurement()

```
returnValue = (36/10) _H_WS1_.windSpeed()
```

HomeMessaging

void sendMesssage(message: String, severity:String)

```
switch severity
  case "LOW"
      _H_EMAIL1.sendEmail("pervmluser@gmail.com",message)
  case "MEDIUM"
      _H_IM1.sendMessage("pervmluser@jabber.org",message)
  case "HIGH"
      _H_SMS1.SMSSender("653092892",message)
  default
      _H_EMAIL1.sendEmail("pervmluser@gmail.com",message))
      _H_IM1.sendMessage("pervmluser@jabber.org",message)
```

StormSecurity

void enable()

```
@enabled = TRUE;
```

void disable()

```
@enabled = FALSE;
```

bool isEnabled()

```
returnValue = @enabled ;
```

void setTreshold(threshold:int)

```
@TheThreshold = threshold;
```

int getTreshold ()

```
returnValue = @TheThreshold ;
```

SittingRoomBlind

void open()

```
if _ST_B1.getPosition() = "CLOSED" then
   _ST_B1.up()
   _ST_B1.up()
else if _ST_B1.getPosition() = "OTHER" then
   _ST_B1.up()
endif
```

void close()

```
if _ST_B1.getPosition() = "OPEN" then
   _ST_B1.down()
   _ST_B1.down()
else if _ST_B1.getPosition() = "OTHER" then
   _ST_B1.down()
endif
```

void middlePosition()

```
if _ST_B1.getPosition() = "CLOSED" then
   _ST_B1.up()
else if _ST_B1.getPosition() = "OPEN" then
   _ST_B1.down()
endif
```

String currentPosition()

```
returnValue = _ST_B1.getPosition()
```

SittingRoomLighting

void switchOn()

```
_ST_GL1_.on()
```

void switchOff()

```
_ST_GL1_.off()
```

bool isLighting()

```
returnValue = _ST_GL1_.getIntensity() > 0
```

void setIntensity(intensity:int)

```
_ST_GL1_.setIntensity(intensity)
```

int getIntensity()

```
returnValue = _ST_GL1_.getIntensity()
```

SittingRoomActivation

bool isActivated()

```
_ST_S1_.isKey1Activated()
```

SittingRoomLightingDetection

int currentLighting()

```
returnValue = (int) _ST_CS1_.lightIntensity()
```

SittingRoomWindow

bool isOpen()

```
returnValue = _ST_CD1_.contactDetectedA()
```

SittingRoomLightingControl

void setTreshold(threshold:int)

```
@TheThreshold = threshold;
```

int getTreshold ()

```
returnValue = @TheThreshold ;
```

FirstFloorAC

void setConfortTemperature(temp:int)

```
@TheTemperature = temp;
```

int getConfortTemperature ()

```
returnValue = @TheTemperature ;
```

FirstFloorTemperature

float currentTemperature()

```
returnValue = (_FF_TS1_.temperature() + _ST_CS1_.temperature())/2
```

GardenLBL

void setTreshold(threshold:int)

```
@TheThreshold = threshold;
```

int getTreshold ()

```
returnValue = @TheThreshold ;
```

GardenLighting

void switchOn()

```
_G_L1_.on();
_G_L2_.on();
```

void switchOff()

```
_G_L1_.off();
_G_L2_.off();
```

bool isLighting()

```
returnValue = ( (_G_L1_.getIntensity() + _G_L2_.getIntensity() ) > 0)
```

void setIntensity(intensity:int)

```
_G_L1_.setIntensity(intensity);
_G_L2_.setIntensity(intensity);
```

int getIntensity()

```
returnValue = _ G_L1_.getIntensity()
```

GardenLightingIntensity

int currentLighting()

```
returnValue = (int) _G_CS1_.lightIntensity();
```

GardenPresence

bool presenceDetected()

```
returnValue =  _G_MD_1_.movementDetected();
```

Debido a que el resto de la especificación funcional del resto de los dispositivos es muy similar a la ya especificada y no aporta ningún valor añadido, se ha optado por no incluirla en la presente documentación

7. Drivers EIB

La integración de los distintos dispositivos y sistemas software que componen el sistema pervasivo se realiza mediante drivers. Los drivers deben permitir acceder desde el framework a la funcionalidad proporcionada por los dispositivos o sistemas software externos. Los drivers para controlar los dispositivos deben lidiar con cuestiones muy específicas de tecnología y fabricante, pero es posible reutilizar los desarrollados en proyectos anteriores si se utiliza el mismo tipo de dispositivo.

7.1. ¿Por qué EIB?

Al iniciar el desarrollo del caso de estudio se planteó la circunstancia de seleccionar la tecnología de control de los dispositivos. Actualmente existen diversas opciones en el mercado (X10, LonWorks, Fagor, etc.), de muy diversas características y cada una con sus virtudes y defectos. Las razones por las que finalmente se optó por la tecnología EIB fueron las siguientes:

- Se trata de un estándar europeo abierto, por lo que no existe una dependencia de un fabricante o empresa en concreto.
- Existe una amplia variedad de dispositivos disponibles que cubrían adecuadamente las necesidades de funcionalidad el caso de estudio.
- Existía en el grupo de investigación personal que conocía la tecnología y que podía formar sobre su funcionamiento.
- Se conocía la existencia de un adaptador de OSGi al bus EIB, por lo que era viable el acceso desde las aplicaciones generadas a los dispositivos.

Pese a ello, no se descarta en un futuro utilizar otra tecnología que también cumple en gran medida estas cualidades.

7.2. Instalación

La instalación realizada para el caso de estudio consta de los siguientes dispositivos:

- Fuente de alimentación de 640 mA y actuador de 6 fases con interfaz RS232 NTA6F16H+COM Serie eibDUO Plus de Lingg & Janke	
- Programador EIBDUOPlus COM12 de ABB	
- Actuador EIB DUO, de 12 canales, 10ª, con manejo manual de Lingg & Janke	

- Dimmer universal de dos canales de Lingg & Janke	
- Actuador de persianas de 4 canales, con manejo manual para carril DIN de Lingg & Janke	
- Termostato continuo de temperatura ambiente (DIGITEMP) de Lingg & Janke	
- Pulsador de placas sensoras Trascent de Merten	
- Pulsador 6232 multifunción con termostato de Merten	
- Pulsador 6227 multifunción de 4 elementos de Merten	
- Sensor combinado de luminosidad y temperatura LU 131 9 200 de Theben	
- Detector de movimiento 180º de JUNG	
- Estación meteorológica 132 9 200 de Theben	
- Sensor de contacto, interfaz de pulsadores 4c. de Linng & Janke	
- Alarma	
- Bombillas	

La instalación de nuestro sistema es centralizada, utiliza un servidor central que nos aporta la expresividad de un lenguaje de programación permitiendo proporcionar una funcionalidad mucho más compleja de la que se puede obtener a partir de una lógica de cableado. El servidor permite un acceso a la funcionalidad de forma homogénea e independiente de las tecnologías de implementación utilizadas abstrayendo al usuario de los detalles de dichas tecnologías, de esta manera, éste solamente invoca operaciones sobre una interfaz. Además, el utilizar un servidor central permite que la lógica de cada dispositivo esté perfectamente encapsulada; por el contrario, en un sistema descentralizado, la lógica se encuentra distribuida por los dispositivos, de manera que el impacto de eliminar un dispositivo no puede ser predicho de una forma intuitiva por el usuario.

Para realizar la instalación física de la red EIB será suficiente con utilizar una línea, puesto que se trata de una vivienda y no de un edificio, y la instalación no sobrepasa los 64 dispositivos. La instalación realizada en el laboratorio se muestra en la siguiente figura.

Figura 15. Entorno real de implantación

Para una mejor visualización de la posición de los dispositivos, se ha realizado un plano de la casa en el que se han distribuido los mismos por los distintos departamentos. Se muestran el primer y segundo piso en las figuras 16 y 17 respectivamente.

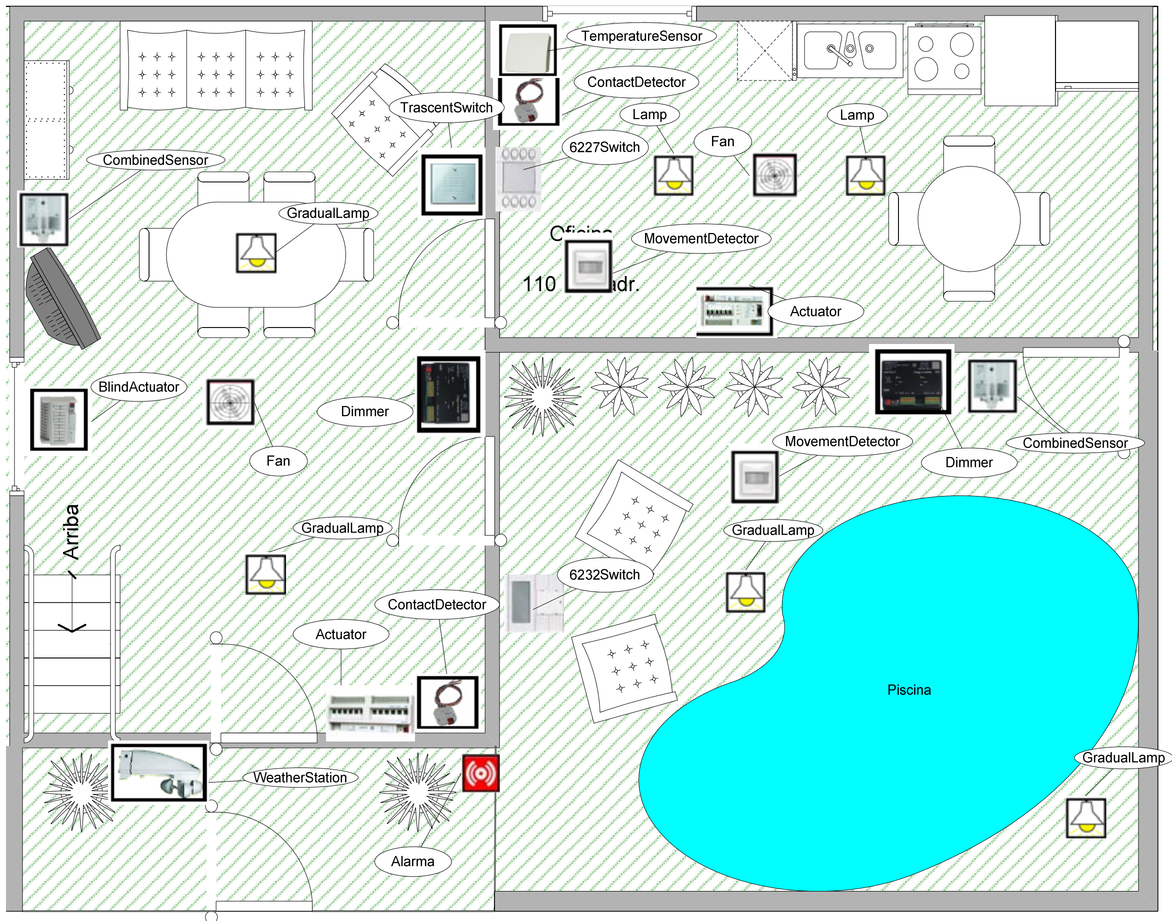


Figura 16. Distribución de los dispositivos en el primer piso

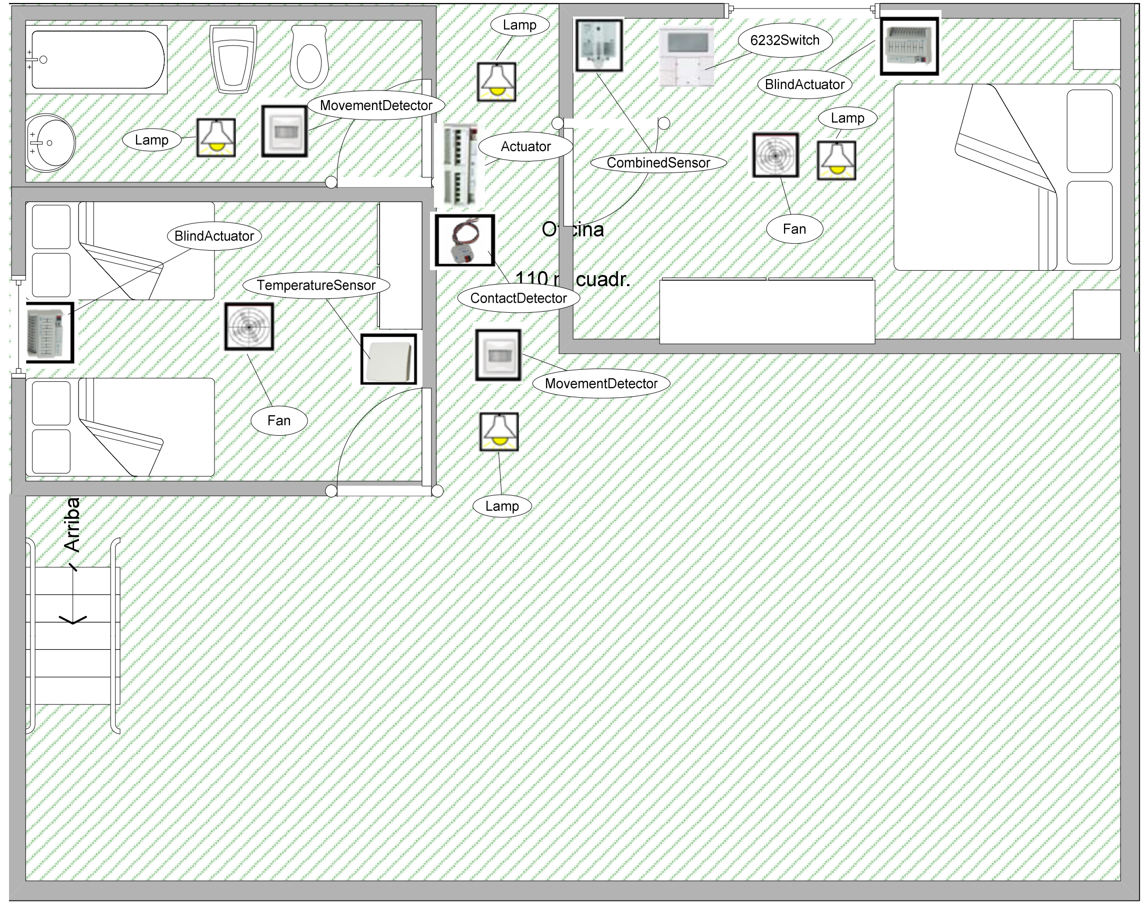


Figura 17. Distribución de los dispositivos en el Segundo piso

La conexión física entre el servidor central y la red EIB se realiza por los puertos serie.

7.3. Configuración mediante ETS2

Para poder configurar el proyecto en el ETS y poder así programar los dispositivos, es necesario, en primer lugar, importar sus paquetes software en el módulo de Administración de Productos. Estos paquetes software son proporcionados, generalmente, por los propios fabricantes de los dispositivos, en ficheros con extensión ".vd1" o ".vd2".

En segundo lugar, debemos diseñar el proyecto. Para ello, en el módulo de Diseño de Proyecto (figura 18), debemos crear las distintas plantas de la casa con sus respectivas habitaciones e insertar todos los elementos de nuestra instalación en las habitaciones correspondientes. Al insertar los dispositivos, el ETS asignará automáticamente una dirección física única para cada uno, aunque ésta puede ser modificada por el usuario siempre y cuando siga siendo unívoca.

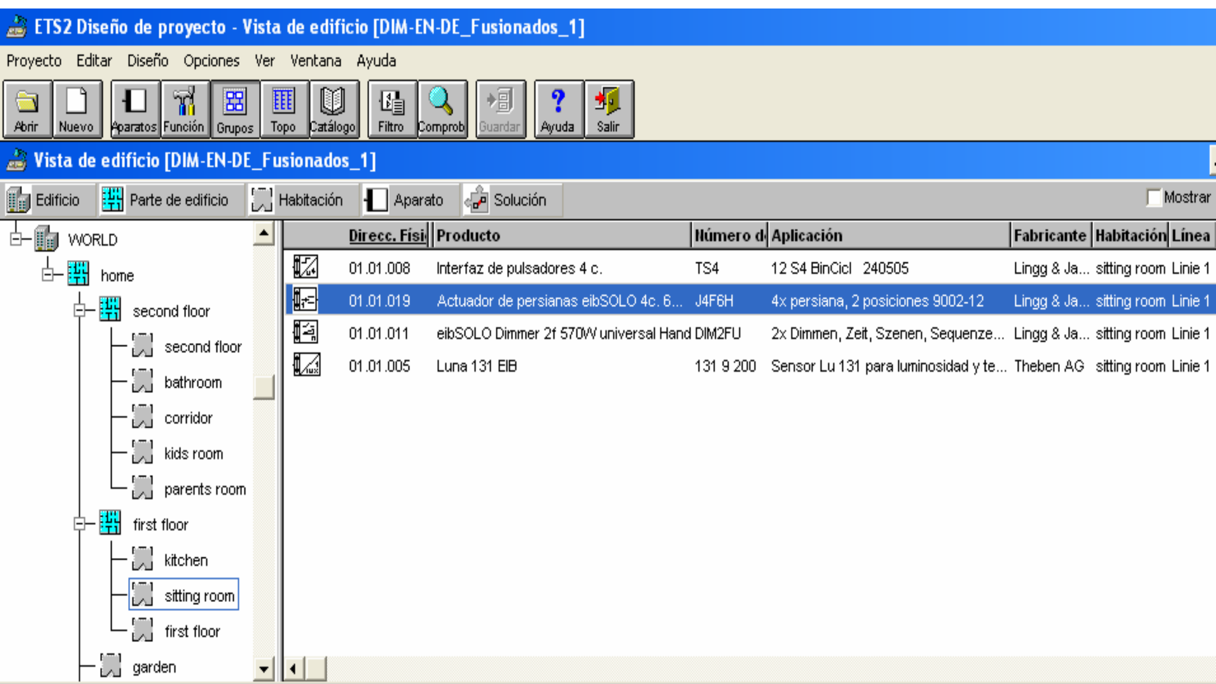


Figura 18. Diseño del proyecto en el ETS

A continuación describiremos la parametrización programada detallada para cada uno de los dispositivos utilizados en la instalación.

Debido a que el caso de estudio es muy extenso, ya que describe la funcionalidad completa que podría ofrecerse en una vivienda real, no se disponía de suficiente cantidad de cada tipo de dispositivo (aunque sí existía al menos un dispositivo de cada tipo), por ello, se optó por realizar la instalación del primer piso y parte del jardín con dispositivos reales, mientras que para la planta superior utilizaremos drivers simulados. Así pues, indicaremos las direcciones de grupo asignadas a los dispositivos del primer piso para poder interactuar con ellos. La asignación de direcciones de grupo para el resto de dispositivos se realizaría de la misma manera.

Las direcciones de grupo son, en la práctica, la forma de clasificar las funciones del sistema. Se ha decidido utilizar direcciones de grupo de tres niveles. El criterio elegido para asignar las direcciones se describe en la figura 19, y se centra en poder localizar los dispositivos fácilmente.

Grupo principal (1er nivel)	Grupo intermedio (2º nivel)	Subgrupo (3er nivel)
0. General 1. Primer piso 2. Segundo piso 3. Jardín	0. Iluminación 1. Seguridad/Alarmas 2. Calefacción/Aire acondicionado 3. Persianas 4. Sensores 5. Interruptores	Referencia al dispositivo en concreto considerando su funcionalidad y su localización en la vivienda.

Figura 19. Criterio para asignar direcciones de grupo

Debemos tener en cuenta que los dispositivos tienen capacidad para funcionar de forma autónoma, sin un control centralizado, por lo que muchas de las funciones que ofrecen son precisamente para que puedan interaccionar entre ellos, proporcionando una funcionalidad a muy bajo nivel. Así pues, parte de la funcionalidad que ofrecen no será necesaria y no se describe a continuación.

1. *Fuente de alimentación y actuador de 6 fases*: La parametrización de este elemento se limita a los actuadores. El dispositivo también permite configurar como debe actuar ante una caída en el bus. Para el caso de estudio hemos utilizado tres de los actuadores. El canal A y el canal B se utiliza para encender o apagar las bombillas de la cocina, y tienen asignadas las direcciones de grupo 1/0/1 y 1/0/2 (primer piso/iluminación/dispositivo), respectivamente, mientras que, el canal C pone en marcha o para un ventilador, la dirección de grupo para interactuar con este canal es 1/2/1 (primer piso/Aire acondicionado/dispositivo). Su dirección física es 1.1.1.

2. *Termostato continuo de temperatura ambiente (DIGITEMP)*: De las funcionalidades que nos ofrece este aparato, la única que nos interesa en este caso es conocer la temperatura actual, por lo que se ha habilitado dicha función y deshabilitado el resto, y se ha asignado al objeto 0 correspondiente a la temperatura actual la dirección de grupo 1/4/1. La dirección física del dispositivo es 1.1.6.

3. *Sensor de contacto TS4*: Este aparato posee 4 canales de manera que cada uno de ellos puede actuar como un sensor de contacto distinto. Debemos asociar una dirección de grupo a los canales que vayamos a utilizar. En la parametrización hemos dejado la configuración por defecto puesto que sólo deseamos que mande un telegrama al bus en el momento en el que cambia su estado.

   En la instalación tenemos dos sensores de contacto, uno situado en el comedor con dirección física 1.1.8, del que sólo usamos un canal cuya dirección de grupo es 1/4/2; y otro situado en la cocina, del que usamos dos canales, con las direcciones de grupo 1/4/3 (sensor de contacto para la puerta) y 1/4/4 (sensor de contacto para la ventana) y cuya dirección física es la 1.1.4.

4. *Dimmer universal de dos canales*: Este dispositivo permite regular la iluminación encendiéndola de una forma gradual hasta alcanzar el valor de iluminación deseado, o bien encenderla o apagarla directamente. Ha sido configurado para que tarde 5 segundos en alcanzar el valor de iluminación indicado, y se le ha asignado una dirección de grupo tanto para el objeto de encendido/apagado, 1/0/3 y 1/0/4, como para el objeto del valor gradual de cada canal, 1/0/5 y 1/0/6. Su dirección física es 1.1.11.

5. *Sensor combinado de luminosidad y temperatura LU 131 9 200*: Este sensor permite establecer hasta 3 umbrales de luminosidad y 2 de temperatura, de manera que se puede configurar para que mande telegramas cuando está por debajo o por encima de cada umbral, o cuando lo supera. Para este caso, nos limitaremos a configurar el dispositivo para poder conocer la temperatura en ºC y la luminosidad en lux actuales. Asignamos pues una dirección de grupo para conocer la luminosidad, 1/4/5 y otro para consultar la temperatura, 1/4/6. La dirección física asignada al dispositivo es la 1.1.5.

6. *Detector de movimiento 180º*: Mediante este detector podemos ser avisados cuando se detecta y se deja de detectar movimiento. Además se puede establecer un retardo de forma que únicamente indique que ha dejado de detectarse movimiento si después de ese retardo se mantiene el estado. El aparato también permite ser bloqueado para que deje de informar de su estado. Ha sido configurado para que envíe un telegrama al bus cuando cambie su estado, estableciendo un retardo de 14,2 seg.

   En la instalación hay dos detectores de movimiento, uno en la cocina, con dirección física 1.1.7 y cuya dirección de grupo es 1/4/7, y otro en el jardín, con dirección física 1.1.2 y cuya dirección de grupo es 3/4/1.

7. *Estación meteorológica 132 9 200*: Posee sensor de luminosidad, temperatura y de lluvia, y mide la velocidad del viento. Permite programar cuando debe enviar telegramas, bien cíclicamente, o bien por cambios de estado de algún sensor, estableciendo las unidades mínimas en las que debe haber variado para que se envíe el telegrama, por ejemplo, cuando la temperatura varíe en 1 ºC. También permite establecer umbrales de luminosidad y temperatura como en el caso de LU 131 9 200. La parametrización actual del dispositivo lo programa para que envíe un telegrama cuando cambie la luminosidad en un 10%, la temperatura en 1 ºC, la velocidad del viento en un 20% y cuando llueva o pare de llover. Las direcciones de grupo asignadas para conocer el valor de cada uno de los sensores nombrados son: 0/4/1 (luminosidad), 0/4/2 (temperatura), 0/4/3 (lluvia), y 0/4/4 (viento). La dirección física del dispositivo es la 1.1.14.

8. *Pulsador de placas sensoras Trascent*: Este pulsador puede programarse para que pueda actuar como una, dos, o 4 placas sensoras independientes. Además, se puede elegir el tipo de datos que envían al accionarse. Se ha configurado para que actúe como 2 placas de tipo EIS1 (1 bit). Las direcciones de grupo asignadas son 1/5/1 y 1/5/2. La dirección física asignada al pulsador es la 1.1.15.

9. *Pulsador multifunción con termostato*: Este dispositivo tiene 4 teclas que actúan como 4 pulsadores independientes. Al igual que el anterior, también se le puede indicar el tipo de datos que deben enviar al bus cuando se

pulsan, y el tipo que se ha seleccionado ha sido EIS1. Su dirección física es 1.1.3, y las direcciones de grupo asignadas para interactuar con los cuatro pulsadores son: 3/5/1, 3/5/2, 3/5/3 y 3/5/4.

10. *Pulsador multifunción de 4 elementos*: Este dispositivo consta de 8 teclas, pero actúan funcionalmente como 4 pulsadores ya que cada par de teclas de la misma altura envía valores contrarios, es decir, considerando las inferiores, si el tipo es EIS1 la de la izquierda enviará un 1 mientras que la de la derecha enviará un 0. Al igual que los pulsadores anteriores, el tipo de datos también se puede seleccionar de una lista de tipos permitidos. Las direcciones de grupo asignadas para interactuar con los cuatro pulsadores son: 1/5/3, 1/5/4, 1/5/5 y 1/5/6. La dirección física asignada al pulsador es la 1.1.16.
11. *Actuador "EIB DUO" de 12 canales*: Este aparato consta de 12 actuadores. Se ha dejado la parametrización por defecto, que indica que un uno lógico es que el actuador está activado, mientras que un 0 lógico indica lo contrario, y que ante una caída del bus, mantenga el valor. También permite asignar a un actuador una función lógica. La dirección física del aparato es la 1.1.10. Solo utilizamos un actuador para activar o desactivar una alarma, por lo que es suficiente con asignarle una dirección de grupo, que es 0/1/1.
12. *Actuador de persianas de 4 canales*: El dispositivo puede controlar hasta 4 persianas, y permite establecer el tiempo que ha de tardar y la cantidad de recorrido que debe hacer la persiana, indicando con un uno lógico subir, y con un cero lógico bajar. Se utiliza para controlar la persiana del salón. La dirección física del actuador es la 1.1.13 y la dirección de grupo asignada es 1/3/1.

El detector de movimiento y los tres tipos de pulsadores descritos necesitan un acoplador de Bus para poder conectarse a la red.

Finalmente, después de realizar la parametrización de los dispositivos, se debe programar ésta (la aplicación) y la dirección física de cada dispositivo en el módulo de Puesta en Marcha (figura 20).

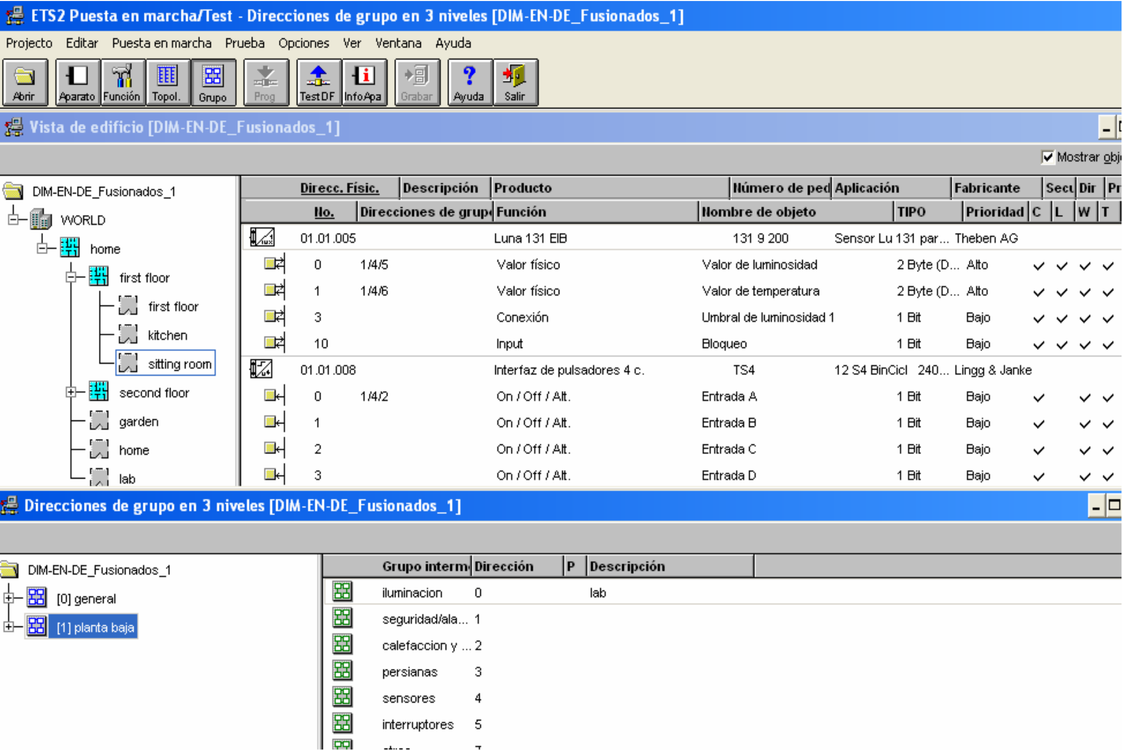


Figura 20. Módulo de puesta en marcha

7.4. Estrategia de Implementación

La programación de los drivers se ha realizado en Eclipse 3.2. Para poder acceder a la red EIB desde OSGi se ha utilizado Prosyst, que es una implementación comercial de OSGI [11]. Prosyst proporciona un paquete EIB que contiene varios bundles para facilitar el acceso a la red.

Los bundles que contiene son:

- *EIB API Bundle*: exporta APIs para controlar los dispositivos de un sistema EIB.
- *EIB Base bundle*: permite encontrar y registrar dispositivos EIB en el framework como si fueran servicios. Además registra diferentes servicios, entre los cuales se encuentra el EIB Group Communicator Service, que permite leer y escribir peticiones en el sistema EIB.
- *PEI16 Driver for Win32*: Implementación de Prosyst para acceder a los drivers a bajo nivel. Permite la conexión al bus a través de la interfaz RS232 de acuerdo con el protocolo de serie PEI 16.
- *EIB admin Bundle*: Proporciona una manera sencilla de visualizar y configurar las propiedades y las direcciones de los dispositivos EIB registrados a través de http.

Los servicios que exportan estos bundles y que utilizaremos para comunicarnos con los dispositivos son: org.mbs.services.eib.EIBGroupComunicator y org.mbs.services.eib.driver.EIBLinkLayer.

Estos servicios nos permiten interactuar con los dispositivos de dos maneras. La primera permite modificar y leer las propiedades o el estado de los dispositivos bajo demanda. Por ejemplo, si deseamos encender una bombilla o saber si está encendida, el servicio org.mbs.services.eib.EIBGroupComunicator nos proporciona métodos para realizar dichas operaciones. En el segundo modo de funcionamiento somos avisados de un cambio. Por ejemplo, si necesitamos conocer cuando un interruptor ha sido pulsado, el servicio org.mbs.services.eib.driver.EIBLinkLayer nos permite suscribirnos a una clase para que se mantenga a la escucha de los posibles cambios que se procesen en el bus.

El primer servicio (org.mbs.services.eib.EIBGroupComunicator) nos permite enviar y recibir telegramas del bus, para lo cual nos proporciona dos métodos:

- Void writeGroupData(int dir_grupo,byte [] datos): este método nos permite enviar un telegrama a la dirección de grupo indicada en el primer parámetro con los datos indicados en el segundo parámetro.

- Byte [] ReadGroupData(int dir_grupo): este método nos permite recuperar la información de la dirección de grupo indicada en el parámetro.

El segundo servicio (org.mbs.services.eib.driver.EIBLinkLayer) nos proporciona el método:

- setLinkLayerListener(EIBLinkLayerListener listener): este método permite que la clase listener pueda observar todos los telegramas que viajan por el bus.

La clase que realiza las funciones listener debe implementar la interfaz EIBLinkLayerListener:

- public void dataIndication(int dir_física_origen, int dir_destino, boolean DAF, int prioridad, byte[] datos): éste método se ejecuta cada vez que se envía un telegrama por el bus, instanciándose los parámetros con la dirección física, la dirección destino, que puede ser una dirección de grupo (DAF=true) o una dirección física (DAF=false), la prioridad del mensaje y los datos.

El servicio org.mbs.services.eib.driver.EIBLinkLayer sólo nos permite utilizar el método setLinkLayerListener para una única clase, por lo que se ha necesitado implemetar una clase auxiliar "Listener", que será la que se suscriba. Esta clase implementará la interfaz producer. De esta manera, todos los drivers que necesiten ser notificados cuando se produzca un cambio en su estado, implementarán la interfaz consumer y crearán un Wire con la clase Listener.

Otro inconveniente es que para poder utilizar los dos servicios (escritura bajo demanda y recepción de notificaciones) al mismo tiempo, es necesario utilizar dos programadores EIB distintos, lo que obliga a que el servidor central tenga dos puertos serie para poder conectar ambos programadores (y, por supuesto, al correspondiente desembolso económico). Esta limitación está impuesta por el módulo proporcionado por Prosyst.

7.5. Ejemplos

Cada driver se implementa mediante dos clases, una clase Activator y otra llamada con el nombre del driver. La clase Activator implementa la interfaz BundleActivator, que consta de dos métodos, start y stop, en los que se indican las acciones que se deben realizar al iniciar o parar el bundle. Mientras que la otra clase implementa los métodos de la interfaz que el driver comparte con el Binding Provider.

Podemos distinguir dos tipos de drivers:

1. los que interactúan con el dispositivo cuando necesitan modificar o leer sus datos,

2. los que necesitan ser notificados cuando el dispositivo ha cambiado.

Así pues, describiremos en esta sección el desarrollo de cada tipo de driver a partir de un ejemplo de cada uno. También describiremos la clase Listener implementada, ya que es necesaria para pueda funcionar más de un dispositivo del segundo tipo al mismo tiempo. En los comentarios de cada clase se describe el funcionamiento detallado de los drivers.

7.5.1. GradualLamp

Este driver regula la intensidad de una bombilla de manera gradual, utilizando el dispositivo Dimmer Universal.

```
public class Activator  implements BundleActivator {
private BundleContext context;
private Dimmer driver;


public void start (BundleContext bc) {
   this.context=bc;
   //Las propiedades que debemos indicar son el PID (indentificador
del servicio)
   //y el tipo de datos que intercambiará a través del wire.

   Properties props = new Properties();
   props.put(org.osgi.framework.Constants.SERVICE_PID,"L1");
   props.put(
org.osgi.service.wireadmin.WireConstants.WIREADMIN_PRODUCER_FLAVORS,
                  new Class[] { HashMap.class });
   driver= new Dimmer(bc);
   try{
     //Registramos el driver como un servicio para cada una de las
interfaces que implementa
      String interfaz=driver.getClass().getInterfaces()[0].getName();
      context.registerService(Producer.class.getName(),driver,props);
      context.registerService(interfaz,driver,props);
   } catch (Exception e) {
      System.out.println("EXCEPTION EIB Dimmer: " + e.getMessage());
   }
}

public void stop (BundleContext bc) {

}
```

```
public class Dimmer implements
org.pervml.bproviders.interfaces.GradualLamp.Interface, Producer{

private BundleContext context;
   private EIBGroupCommunicator eib;
private int fromEIS1(byte [] data)//Transformamos los datos de tipo
EIS6 en un entero
{
      int eis1=data[0];
      return eis1;
}
private byte [] toEIS1(int data)//Transformamos los datos a tipo
EIS6 para poder enviarlos
{
      byte[] datos=new byte[1];
      datos[0]=(byte)data;
      return datos;
}
//Transformamos los datos de tipo EIS6 en un entero
public int fromEIS6(byte[] data) {
      int datos=data[0];
      return datos;
}
//Transformamos los datos a tipo EIS6 para poder enviarlos
public byte[] toEIS6(int data) {
      byte [] value=new byte[2];
      value[0]=0x00;
      value[1]=(byte) data;
      return value;
}
//Recuperamos EIBgroup comunicator
private void connectToEIBbus(){
   //Busca el servicio EIBGroupCommunicator para
   //poder leer y enviar información al driver.

      ServiceTracker tracker=new ServiceTracker(context,
      org.mbs.services.eib.EIBGroupCommunicator.class.getName(),
      null);
      tracker.open();
      Object [] services = tracker.getServices();
      if (!(services == null))
      {

         if (services[0].getClass().getName().equals(
            "com.prosyst.mbs.impl.framework.ServiceReferenceImpl"))
                  this.eib = (EIBGroupCommunicator)
            this.context.getService((ServiceReference) services[0]);
         else
            this.eib = (EIBGroupCommunicator) services[0];
      }
}
   //Fin Recuperamos EIBgroup comunicator
```

```
//Interfaz producer
protected Wire [] wiresConsumer = null;

public void consumersConnected(Wire [] ws) {
      this.wiresConsumer = ws;
}
public Object polled(Wire w) {
      return null;
}
protected void notifyConsumers() {
      if ( this.wiresConsumer == null ) return;
      for (int i=0; i<this.wiresConsumer.length; i++){
            this.wiresConsumer[i].update(null);
      }
}
//Interfaz producer END
public Dimmer  (BundleContext bc) {
      context = bc;
      //llamada para recuperar eibgroup comunicator
      connectToEIBbus();
}

//metodos de la interfaz
public void on()
  {
      try{
            byte [] value=new byte[1];
            value[0]=1;
      //Escribimos un 1 en la dirección de grupo correspondiente
      //para encender la bombilla
            eib.writeGroupData ( 2307, value );
            }
      catch(Exception e){
            System.out.println("EXCEPTION EIB");
      }
}
public void off()
{
      try{
            byte [] value=new byte[1];
            value[0]=0;
      //Escribimos un 0 en la dirección de grupo correspondiente
      //para apagar la bombilla
            eib.writeGroupData ( 2308,  toEIS6(0) );
                  }
      catch(Exception e){
            System.out.println("EXCEPTION EIB");
      }
}
```

```java
public boolean isOn() {
	int data=0;

	try{
		//Comprobamos si la bombilla está encendida
		data= fromEIS6(eib.readGroupData ( 2307));
	}
	catch(Exception e){
		System.out.println("EXCEPTION EIB");

	}
	if(data==1)return true;
	else return false;
}

public void setIntensity(int intensity) {

	try{
		int aux=(255*intensity)/100;
		if(aux<=255 && aux>=0)
		//Escribimos la intensidad de iluminación en la
		//dirección de grupo correspondiente
			eib.writeGroupData ( 2308,  toEIS6(aux) );
		else System.out.println("La intensidad no es un número
comprendido entre 0 y 100");

	}
	catch(Exception e){
		System.out.println("EXCEPTION EIB");
	}
}

public int getIntensity() {
	int data=0;
	int intensity=0;
	try{
		//Leemos la intensidad de ilumninación de la
		//dirección de grupo correspondiente
		data= fromEIS6(eib.readGroupData(2308));
		intensity=(100*data)/255;
			}
	catch(Exception e){
		System.out.println("EXCEPTION EIB");}
	return intensity;

}
```

7.5.2. WallSwitch

Este driver se utiliza para controlar pulsador de placas sensoras Trascent.

```
public class Activator  implements BundleActivator {
       private BundleContext context;
       private WallSwitch switchEIB;
       private WireAdmin wa;

public void start (BundleContext bc) {
   this.context=bc;
   //Las propiedades que debemos indicar son el PID (indentificador
   del servicio)
   //y el tipo de datos que intercambiará a través de sus wires.
   String driverPID="trascent";
   Properties props = new Properties();
   props.put(org.osgi.framework.Constants.SERVICE_PID,driverPID);
   props.put(
org.osgi.service.wireadmin.WireConstants.WIREADMIN_PRODUCER_FLAVORS,
            new Class[] { HashMap.class });
   props.put(
org.osgi.service.wireadmin.WireConstants.WIREADMIN_CONSUMER_FLAVORS,
            new Class[]{HashMap.class} );
   try{
     switchEIB= new WallSwitch(bc);
     String interfaz=
                 switchEIB.getClass().getInterfaces()[0].getName();
     //Registramos el driver como un servicio para cada una de las
interfaces que implementa
     context.registerService(Consumer.class.getName(),switchEIB,props);
     context.registerService(Producer.class.getName(),switchEIB,props);
     context.registerService(interfaz,switchEIB,props);
     ServiceReference sr =
            context.getServiceReference(WireAdmin.class.getName());
     if (sr!= null) {
          wa = (WireAdmin) context.getService(sr);}
     String producerPID = "Listener";
     String wid = producerPID+driverPID;
     Hashtable props2 = new Hashtable();
     props.put("PervML_WireID",wid);
     //Creamos el wire con la clase Listener
     wa.createWire(producerPID, driverPID, props2);
   } catch (Exception e) {
      System.out.println("\n\n\nexception: " + e.getMessage());}
}
public void stop (BundleContext bc) {
      Wire aux;
      try{ //Eliminamos los wires enlazados con el driver
         Wire [] wires=wa.getWires(null);
         for(int i=0;i<wires.length;i++){
            aux= (Wire) wires[i];
            wa.deleteWire(aux);}
        }
        catch(Exception E){System.out.println("excepcion al eliminar
los wires" + E);}
}
}
```

```java
public class WallSwitch implements
org.pervml.bproviders.interfaces.Switch.Interface, Consumer,
Producer {

private boolean key1 = false;
private boolean key2 = false;
private boolean key3 = false;
private boolean key4 = false;
private BundleContext context;

//Interfaz Consumer
protected Wire [] wiresProducer = null;
	public void producersConnected(Wire[] wp){
				System.out.print("funciona consumers
conected"+ wp.length);
			this.wiresProducer = wp;
			System.out.println("EIB driver cantidad
productores: "+wiresProducer.length);


	}
public void updated(Wire arg0, Object arg1) {
	//Éste método se ejecutará cada vez que un telegrama viaje por
el bus
	//Comprobamos que la dirección física es la correspondiente al
wallSwitchTrascent
	if(dir_fisica==4367){ //Merten en la parte inferior izq
		if( dir_grupo==19){	//arriba izquierda
			if(encendido==1){
				key1=true;
			}
			else if(encendido==0){key1=false;
							}
		else if(dir_grupo==16){ //arriba derecha
			if(encendido==1)
				key2=true;
			else key2=false;
			}
		else if(dir_grupo==20){ //inferior izquierda
			if(encendido==1)
				key3=true;
			else key3=false;
		}
		else if(dir_grupo==18){ //inferior derecha
			if(encendido==1)
				key4=true;
			else key4=false;
		}
	}
}
```

```
//Interfaz Producer
protected void notifyConsumers() {
    if ( this.wiresConsumer == null ) return;
    for (int i=0; i<this.wiresConsumer.length; i++){
        if(!(wiresConsumer[i].getProperties().get(
                "wireadmin.consumer.pid").equals("")))
            {  this.wiresConsumer[i].update(null);
            }
    }
}

protected Wire [] wiresConsumer = null;
public void consumersConnected(Wire [] ws) {
    this.wiresConsumer = ws;
}
public Object polled(Wire w) {
    return null;
}
//Fin Interfaz Producer


public WallSwitch (BundleContext bc) {
        context = bc;
}


    //Interfaz del driver
public boolean isKey1Activated() {
        return key1;
}
public boolean isKey2Activated() {
        return key2;
}
public boolean isKey3Activated() {
        return key3;
}
public boolean isKey4Activated() {
        return key4;
}
```

7.5.3. Listener

```
public class Activator  implements BundleActivator {
private BundleContext context;
private Listener listener;
EIBLinkLayer eibDUO;
private void connectToEIBbus(){
   ServiceTracker tracker=new ServiceTracker(context,
      org.mbs.services.eib.driver.EIBLinkLayer.class.getName(),
      null);
   tracker.open();
   Object [] services = tracker.getServices();
   if (!(services == null))
      {ServiceReference r;
      if (services[0].getClass().getName().equals(
            "com.prosyst.mbs.impl.framework.ServiceReferenceImpl"))
            {//Si nos dan la referencia del servicio, creamos la
               Instancia del servicio
            r=(ServiceReference) services[0];
            if(r.getProperty("PORT").equals("COM2")){
                  this.eibDUO = (EIBLinkLayer)
                  this.context.getService(
                              (ServiceReference) services[0]);
            }
      else
      if (services[1].getClass().getName().equals(
            "com.prosyst.mbs.impl.framework.ServiceReferenceImpl"))
            {//Si nos dan la referencia del servicio, creamos la
               Instancia del servicio
            r=(ServiceReference) services[1];
            if(r.getProperty("PORT").equals("COM2")){
                  this.eibDUO = (EIBLinkLayer)
                              this.context.getService(
                              (ServiceReference) services[1]);
            }
      }
   }
}
```

```
public void start (BundleContext bc) {

this.context=bc;

//Las propiedades que debemos indicar son el PID (indentificador del
servicio) y el tipo de datos que intercambiará a través del wire.
Properties props = new Properties();
props.put(org.osgi.framework.Constants.SERVICE_PID,"Listener");
props.put(
org.osgi.service.wireadmin.WireConstants.WIREADMIN_PRODUCER_FLAVORS,
   new Class[] { HashMap.class });

try{
   this.connectToEIBbus();

   listener= new Listener(bc);

   String interfaz =listener.getClass().getInterfaces()[0].getName();
   String interfazEIB
            =listener.getClass().getInterfaces()[1].getName();
   context.registerService(Producer.class.getName(),listener,props);
   context.registerService(interfaz,listener,props);
   eibDUO.setLinkLayerListener(listener);

} catch (Exception e) {
      System.out.println("Exception en Listener: " + e.getMessage());
}

}


public void stop (BundleContext bc) {}

}
```

```
public class Listener implements EIBLinkLayerListener, Producer {

private BundleContext context;
public Listener (BundleContext bc) {
  this.context = bc;
}

//Interfaz Producer
protected void notifyConsumers() {
  if ( this.wiresConsumer == null ) return;

  for (int i=0; i<this.wiresConsumer.length; i++){
  if(!(wiresConsumer[i].getProperties().get(
        "wireadmin.consumer.pid").equals("")))
    {this.wiresConsumer[i].update(null);
     wiresConsumer[i].getProperties().get("wireadmin.producer.pid");
     }
  }
}

protected Wire [] wiresConsumer = null;
public void consumersConnected(Wire [] ws) {
  this.wiresConsumer = ws;

}
public Object polled(Wire w) {
  return null;
}
//Fin Interfaz Producer


//Interfaz EIBLinkLayerListener

//dir fisica origen,dir de grupo->destino, DAF,prioridad, datos
public void dataIndication(int arg0, int arg1, boolean arg2, int
arg3, byte[] arg4) {
      for (int i = 0; wiresConsumer != null && i <
wiresConsumer.length; i++)
         {
         HashMap h=new HashMap();
         h.put("dir_fisica", Integer.valueOf(arg0));
         h.put("dir_grupo", Integer.valueOf(arg1));
         h.put("datos", arg4);

         //Notificamos a todos los consumidores, pasándoles los
datos necesarios
         wiresConsumer[i].update(h);
                             }

}
//Fin Interfaz EIBLinkLayerListener

}
```

8. Drivers simulados

En el caso de estudio realizado no se dispone de todos los dispositivos necesarios para realizar la instalación completa de la vivienda, por lo que se han utilizado distintos drivers virtuales para poder probar el sistema final.

Para realizar la implementación de los dispositivos virtuales, se ha utilizado un framework [12], desarrollado anteriormente en otro Proyecto Final de Carrera, para facilitar la simulación de sistemas pervasivos. El framework ha sido desarrollado sobre la tecnología OSGi y proporciona mecanismos para el desarrollo de dispositivos virtuales, y su monitorización y control mediante una interfaz gráfica de usuario.

Se introduce en primer lugar cual es la estructura del framework y como se utiliza, para posteriormente describir los drivers simulados que han sido desarrollados específicamente para este caso de estudio.

8.1. Descripción del framework para drivers simulados

8.1.1. Estructura del framework

El framework para la simulación de sistemas pervasivos está estructurado en tres partes bien diferenciadas: la infraestructura de clases abstractas para la creación de dispositivos virtuales, un catálogo de dispositivos virtuales creados utilizando la infraestructura anterior, y una interfaz de usuario para poder interactuar con los dispositivos virtuales cuando estos se encuentran en ejecución. Antes de pasar a describir cada una de estas partes, vamos a presentar el esquema global de funcionamiento que sigue el framework.

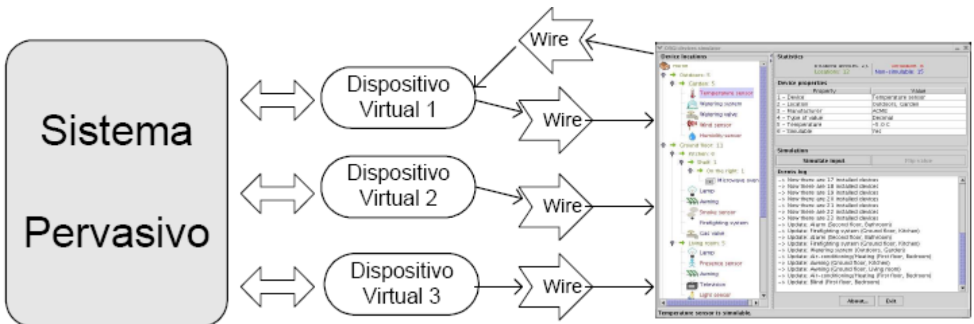


Figura 21. Estructuración del framework

Tal y como se presenta en la figura 21, los dispositivos virtuales serán utilizados por el sistema pervasivo, pero no se impone ninguna restricción sobre cómo se realizará esta comunicación. De este modo se consigue que nuestro framework sea independiente del método utilizado para desarrollar el sistema pervasivo. Por otra parte, la comunicación entre los dispositivos y la interfaz de usuario del simulador se realiza utilizando los mecanismos de comunicación que proporciona OSGi llamados

Wires. A través de este mecanismo se envían paquetes de propiedades (HashMap) en ambas direcciones. De este modo se facilita un alto desacoplamiento entre los dispositivos y la interfaz, ya que no necesitan referencias mutuas explícitas. A continuación se describen detalladamente los componentes del framework.

8.1.2. Infraestructura para el desarrollo de dispositivos virtuales

La infraestructura que proporciona el framework para el desarrollo de dispositivos, que se muestra en la figura 22 se compone de: clases abstractas para la creación de dispositivos y una clase auxiliar para la gestión de estos dispositivos.

El framework proporciona dos clases abstractas llamadas VirtualDevice y Virtual- ManagedDevice. Todos los dispositivos virtuales deben extender una de estas dos clases.

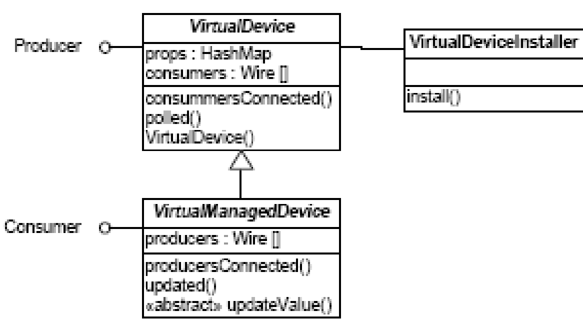


Figura 22. Clases de la infraestructura software del framework.

La clase abstracta VirtualDevice se encarga de:

- Definir un atributo llamado props de tipo HashMap para almacenar las propiedades del dispositivo.
- Definir una serie de constantes que serán utilizadas para referenciar a las propiedades. Por ejemplo, se definen propiedades para especificar el tipo de dispositivo, la ubicación, el fabricante, el tipo de valor que almacenará la propiedad, etc. En la sección 5 se comentarán detalladamente estas propiedades.
- Definir un constructor por defecto, el cual recibe como parámetro la ubicación del dispositivo. Las clases que extiendan de VirtualDevice deberán sobrescribir este constructor.
- Implementar la interfaz que le permite participar como emisor de información (Producer) en el mecanismo de comunicaciones de OSGi.

Por su parte, la clase VirtualManagedDevice es una especialización de la clase anterior. Esta clase está ideada para aquellos dispositivos sensores que reciben información del entorno, de manera que los valores de sus propiedades podrán fijarse desde la interfaz del simulador. Además de lo especificado para la clase VirtualDevice, la clase VirtualManagedDevice se encarga de:

- Implementar la interfaz que le permite participar como receptor de información (Consumer) en el mecanismo de comunicaciones de OSGi.

- Definir la operación abstracta updateValue que deberá ser implementada en las clases concretas para modificar adecuadamente el estado del dispositivo a partir de los valores proporcionados por el usuario.

Finalmente, la clase estática VirtualDeviceInstaller se encarga de realizar las acciones necesarias para instalar adecuadamente un nuevo dispositivo virtual en el entorno. Esta clase tiene un método llamado install que recibe como parámetros una instancia del dispositivo que se desea instalar y un nombre, a partir de los cuales:

- registra al dispositivo virtual como generador (y productor, si es necesario) de información.

- crea canales de comunicación (Wire) entre el dispositivo y la interfaz de usuario (un canal para enviar información y otro, si es necesario, para recibir información de la interfaz).

8.1.3. Uso del framework

Esta sección describe como realizar dos tareas importantes para hacer uso del framework: la creación de nuevos tipos de dispositivos y la creación de instancias de esos tipos en el sistema.

8.1.3.1. Creación de nuevos tipos de dispositivos

Para la creación un nuevo tipo de dispositivo virtual se debe crear una nueva clase Java, siguiendo el siguiente procedimiento:

1. En primer lugar hay que decidir si el nuevo tipo de dispositivo tendrá alguna propiedad que ha de ser controlada desde la interfaz de usuario. Si es así la nueva clase heredará de la clase abstracta VirtualManagedDevice. En caso contrario deberá heredar de la clase VirtualDevice. Independientemente de esto, la clase Java podrá implementar tantas interfaces como sean necesarias, ya que esto no entrará en conflicto con su funcionamiento como dispositivo simulado.

2. El constructor de la clase deberá: (1) llamar al constructor de la clase padre (this.super()) utilizando como argumento la cadena de texto con la ubicación del dispositivo virtual, (2) insertar en la variable props (que es de tipo HashMap) las propiedades necesarias, y (3) realizar otras taréas no relacionadas con la simulación del dispositivo. Respecto a las propiedades a insertar en la variable props cabe destacar que algunas de ellas son obligatorias para el correcto funcionamiento del mecanismo de simulación de dispositivos:

- DEVICE: almacena el nombre del dispositivo.
- LOCATION: almacena la ubicación del dispositivo. Esta propiedad es añadida por el constructor de la clase abstracta.
- VALUETYPE: almacena el tipo del valor que gestiona el dispositivo. Los posibles valores de esta propiedad son NUMERIC, BOOLEAN, STRING o MULTIPLE. La interfaz del simulador utiliza esta propiedad para proporcionar el mecanismo más adecuado para editar el valor.
- EDITABLE: indica si el valor que gestiona el dispositivo puede ser modificado por el usuario (si se trata de algún tipo de sensor).
- IMAGE: contiene la ruta del icono que se debe utilizar en el árbol de dispositivos para representar a este dispositivo.

El resto de propiedades son mostradas al usuario sin que tengan ninguna implicación en el funcionamiento del sistema.

3. En cualquier momento de la ejecución de la clase que representa el dispositivo simulado es posible realizar la comunicación con el simulador para informar de alguna incidencia. Para ello será necesario seguir los siguientes pasos: (1) crear una variable de tipo HashMap, (2) insertar las propiedades cuyo valor deba ser modificado y (3) transmitir el diccionario (para lo cual será necesario invocar el método update de los canales de comunicación: consumers[i].update(h)). Además, el simulador proporciona la capacidad de enviar mensajes al usuario. Para ello será necesario incluir en el diccionario una propiedad llamada MESSAGE. El contenido de esta variable será mostrado al usuario en una ventaja emergente.

4. En el caso de que exista alguna propiedad que pueda ser modificada (en cuyo caso se habrá extendido la clase VirtualManagedDevice), será necesario implementar el método updateValue. Este método recibe como parámetro una cadena de texto que deberá ser convertida al tipo de datos adecuado antes de almacenar su valor y/o realizar las acciones oportunas.

5. Finalmente, se implementará el resto de la clase siguiendo las directrices que dicte el sistema pervasivo con el que interactuará el dispositivo virtual.

8.1.3.2. Creación de una nueva instancia de un tipo de dispositivo

La creación de una nueva instancia de dispositivo se debe realizar desde una clase Java. Un modo habitual de proceder, es utilizar una clase para instalar todos los dispositivos que necesitará utilizar un sistema. Para instalar un dispositivo se deben realizar los siguientes pasos:

1. Crear una nueva instancia de la clase Java que implementa el dispositivo virtual que queremos instalar. Será necesario crear tantas instancias como dispositivos virtuales deseen instalarse en el sistema.

2. Registrar adecuadamente el dispositivo en el entorno. Para ello se utilizará el método install de la clase VirtualDeviceInstaller. Este método recibe como parámetros la instancia creada en el paso anterior, un nombre y un identificador único en el entorno.

El siguiente trozo de código muestra un ejemplo de los pasos descritos:

```
VirtualDeviceInstaller devInst =
    new VirtualDeviceInstaller();
    LampDevice lamp001 = new LampDevice("/Home/Corridor:001");
    devInst.install(lamp001, "Corridor Lamp", "org.oomethod.lamp001");
```

8.2. Ejemplos de drivers desarrollados

Los drivers que se han desarrollado en el presente trabajo son los que simulan cada uno de los dispositivos EIB utilizados, además de los drivers multimedia descritos y del driver para la obtención de información meteorológica. En lo que respecta a la implementación podemos distinguir dos tipos de drivers virtuales, los que necesitan controlar alguna propiedad, es decir, los que pueden modificar su estado, como por ejemplo un interruptor, que puede ser activado o desactivado físicamente, o un sensor de temperatura, cuya temperatura puede variar; y los que no necesitan controlar ninguna propiedad, como por ejemplo la iluminación. Explicamos a continuación un ejemplo de cada uno de los tipos nombrados.

8.2.1. GradualLamp

Las iluminación gradual no puede controlarse desde la interfaz del simulador.

```
package org.pervml.drivers.gradualLamp;

import org.pervml.devicesimulator.installation.HomeInstaller;

public class Activator extends HomeInstaller  {

      public void doHomeInstall () {
            //Creamos las instancias para la iluminación gradual
            //de la casa que ha de hacerse con drivers virtuales
            installDevice(new GradualLamp("Home/Garden"),"G_GL1");
            installDevice(new GradualLamp("Home/Garden"),"G_GL2");



}
```

```
package org.pervml.drivers.gradualLamp;

import java.util.HashMap;
import org.pervml.bproviders.interfaces.gradualLamp.Interface;
import org.pervml.devicesimulator.devices.VirtualDevice;

public class GradualLamp extends VirtualDevice implements
org.pervml.bproviders.interfaces.gradualLamp.Interface {

      private int intensity = 0;

      public GradualLamp (String u) {
            super(u);
            props.put(DEVICE, "GradualLamp");
            props.put(INTENSITY, "0");
            props.put(SIMULABLE, FALSE);
            props.put(VALUE_TYPE, NUMERIC);
            props.put(IMAGE, "lamp_off.gif");
            }
```

```
	public void On () {
		if (intensity!=100)
		{intensity = 100;
		updateWires();
		}
	}
	public void Off () {
		if (intensity!=0)
		{intensity = 0;
		updateWires();
		}
	}
	public boolean isOn(){
		if (intensity>0) return true;
			else return false;
	}
	public void setIntensity(intensidad)() {
		intensity=intensidad;
		updateWires();

	}

	public int getIntensity() {
		return intensity;

	}

	private void updateWires () {
		props.put(INTENSITY, String.valueOf(intensity));
		props.put(IMAGE, ((intensity > 0) ? "lamp_on.gif" :
"lamp_off.gif"));
		HashMap h = new HashMap();
		props.put(INTENSITY, String.valueOf(intensity));
		h.put(IMAGE, ((intensity > 0) ? "lamp_on.gif" :
"lamp_off.gif"));
		super.updateWires(h);
	}
}
```

8.2.2. WallSwitch

El interruptor puede ser activado o desactivado desde la interfaz del simulador.

```
package org.pervml.drivers.wallSwitch;



import org.pervml.devicesimulator.installation.HomeInstaller;

public class Activator extends HomeInstaller  {
      public void doHomeInstall () {

      //Creamos las instancias para los interruptores
      //de la casa que han de hacerse con drivers virtuales
      installDevice(new WallSwitch("Home/Parents_Room"),"PR_S1");

      }
}
```

```java
package org.pervml.drivers.wallSwitch;

import java.util.HashMap;
import org.pervml.bproviders.interfaces.wallSwitch.Interface;
import org.pervml.devicesimulator.devices.VirtualDevice;
import org.pervml.devicesimulator.devices.VirtualManagedDevice;
import java.util.HashMap;

public class WallSwitch extends VirtualManagedDevice implements
org.pervml.bproviders.interfaces.wallSwitch.Interface {

	private boolean State = false;

	public WallSwitch (String u) {
		super(u);
		props.put(DEVICE, "wallSwitch");
		props.put("STATE", "OFF");
		props.put(SIMULABLE, TRUE);
		props.put(VALUE_TYPE, YES_NO);
		props.put(IMAGE, "power_off.gif");
	}

	public boolean activated() {
		System.out.println("getState driver" + State);
		return State; }

	protected void updateValue (String value) {

		State=!State;
		props.put("STATE", (State ? "ON" : "OFF"));
		props.put(IMAGE, (State ? "power_on.gif" :
"power_off.gif"));
		HashMap h = new HashMap();
		h.put("STATE", (State ? "ON" : "OFF"));
		h.put(IMAGE, (State? "power_on.gif" : "power_off.gif"));
		h.put(LOGGABLE, new Boolean(false));
		super.updateWires(h);
	}
}
```

9. Drivers software: SMS

Para desarrollar éste driver se ha utilizado el paquete SMS que proporciona Prosyst. Este paquete permite la gestión de la recepción y el envío de SMS (Short Message Service) a través de GSM (Global System for Mobile Communication).

El paquete proporciona dos servicios: SMS Service, que permite la recepción y envío de mensajes SMS a través de un centro SMS, y ManagedService, que permite la configuración del servico SMS.

El SMS Service implementa dos interfaces:

- com.prosyst.mbs.services.sms.SMS:
- void addNMIListener(NMIListener listener): suscribe a la clase listener para que sea notificada cuando se reciba un mensaje.
- void removeNMIListener(NMIListener listener): detiene las notificaciones a la clase listener por la recepción de nuevos mensajes.
- Void Send(SMSMessage msg): envía un nuevo mensaje SMS. En msg se indican el tipo del mensaje, la dirección destino y el texto del mensaje.
- com.prosyst.mbs.services.sms.NMIListener:
- void messageReceived(SMSMessage msg): este método es llamado cuando se recibe un mensaje.

Mediante el servicio ManagedService configuramos desde prosyst el servicio SMS. Debemos tener en cuenta que el móvil se conecta al ordenador mediante bluetooth, ya que el sistema operativo trata las conexiones bluetooth como si fueran puertos COM. En la figura 23 se muestran las propiedades que deben configurarse.

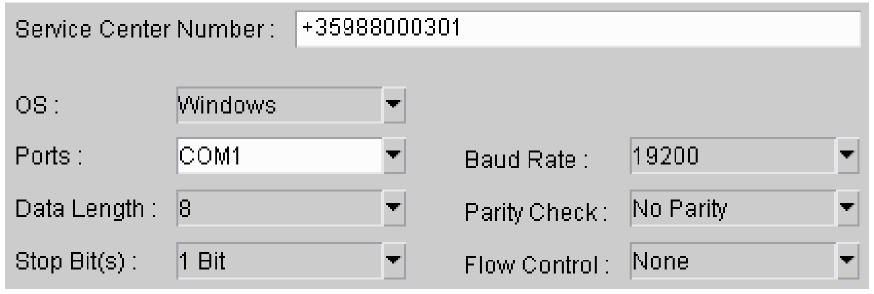


Figura 23. Configuración de las propiedades SMS.

Se describe a continuación la implementación del driver.

```
public class Activator  implements BundleActivator {

  private BundleContext context;
  private ArrayList wires = new ArrayList();
  private WireAdmin wa;
  public void start (BundleContext bc) {
  this.context=bc;
  try{
     SMSdriver smsDriver= new SMSdriver(bc);
   //Las propiedades que debemos indicar son el PID (indentificador
del servicio)
   //y el tipo de datos que intercambiará a través del wire.
     Properties props = new Properties();
     props.put(org.osgi.framework.Constants.SERVICE_PID,"PERVML-SMS");
     props.put(
  org.osgi.service.wireadmin.WireConstants.WIREADMIN_PRODUCER_FLAVORS,
                 new Class[] { HashMap.class });

     //Registramos el driver como un servicio para cada una de las
interfaces que implementa
     String interfaz =SMSdriverI.class.getName();
     context.registerService(Producer.class.getName(),smsDriver,props);
     context.registerService(interfaz,smsDriver,props);
     } catch (Exception e) {
        System.out.println("DRIVER SMS: EXCEPTION! " + e.getMessage());
            }
      }

 public void stop (BundleContext bc) {}


}
```

```
public class SMSdriver implements SMSdriverI, Producer, NMIListener {

private String numberPrefix="+34";
private SMS smsService;
private ArrayList allSMSmsg;
private ArrayList noProcessedSMSmsg;
private SMSMessage lastSMSmsg;

private BundleContext context;
public SMSdriver (BundleContext bc) {
	allSMSmsg=new ArrayList();
	context = bc;
	connectToSMSservice();
	this.smsService.addNMIListener(this);
	}

protected Wire [] wiresConsumer = null;

//Métodos de Producer
public void consumersConnected(Wire [] ws)
{this.wiresConsumer = ws;
}

public Object polled(Wire w)
{	return null;
}

protected void notifyConsumers() {
	if ( this.wiresConsumer == null ) return;
	for (int i=0; i<this.wiresConsumer.length; i++){
	  if(!(wiresConsumer[i].getProperties().get(
	       "wireadmin.consumer.pid").equals("")))
	  {this.wiresConsumer[i].update(null);
	  wiresConsumer[i].getProperties().get("wireadmin.producer.pid");
	  }
	}
}
//Fin métodos de Producer
```

```
private void connectToSMSservice(){
    //Buscamos el servicio SMS para poder enviar y recibir mensajes
    ServiceTracker tracker=new
          ServiceTracker(context,SMS.class.getName(),null);
    tracker.open();
    Object [] services = tracker.getServices();
    if (!(services == null))
    {  //Si es una referencia creamos la instancia del servicio
       if(services[0].getClass().getName().equals(
             "com.prosyst.mbs.impl.framework.ServiceReferenceImpl"))
          this.smsService = (SMS)
             this.context.getService((ServiceReference) services[0]);
       else
             //Si es el servicio, hacemos el casting
          this.smsService = (SMS) services[0];
    }
}
public void sendSMS(String number, String text){
    try{
    //Antes de enviar, comprobamos que tenemos el servicio SMS, sino,
debemos buscarlo
    if(smsService==null) this.connectToSMSservice();
    if (!number.startsWith("+")) number=this.numberPrefix+number;

    //SMSMessage(byte msgType, java.lang.String address, java.lang.String data),
       SMS_SUMIT indica que es un mensaje para enviar.
    SMSMessage smsparaEnviar= new
                         SMSMessage(SMSMessage.SMS_SUBMIT,number,text);
    smsService.send(smsparaEnviar);
    }
    catch(Exception e){
       System.out.println("SMS: fallo al enviar sms: "+ e.getMessage());
    }
}
//Devuelve el último mensaje recibido y lo elimina de la lista de
//mensajes no procesados
public String getLastSMSmsgText(){
       this.noProcessedSMSmsg.remove(this.lastSMSmsg);
       return this.lastSMSmsg.getData().toString();
}
public ArrayList getListSMSmsgText(){
       return this.allSMSmsg;
}
//Devuelve la lista de mensajes
public ArrayList getListSMSmsgText(){
       return this.allSMSmsg;
}

//métodos NMIListener
public void messageReceived(SMSMessage msg) {
       this.allSMSmsg.add(msg);
       this.lastSMSmsg=msg;
this.noProcessedSMSmsg.add(msg);

       this.notifyConsumers();
}//fin métodos NMIListener
}
```

10. Interfaces de usuario

Para interactuar con el sistema pervasivo, se ofrecen dos interfaces de usuario. Ambas son interfaces web: una para acceder desde navegadores de escritorio (figura 26) y otra para acceder desde navegadores de dispositivos móviles (figura 25).

10.1. Interfaz para PDA

La aproximación para desarrollar la web para PDA es utilizar un motor de plantillas (templates engines). El que se ha utilizado es Freemarker. FreeMarker es un motor de plantillas de Java diseñado para generar páginas web HTML, particularmente para aplicaciones basadas en servlet que siguen el patrón MVC. Genera una página web, tal y como indica la figura 24 a partir de una estructura java, donde almacenaremos los datos del sistema, y una plantilla, que se aplicará sobre los datos para generar la página final. Así pues, deberemos tener una plantilla por cada página diferente que puede visualizar el usuario.

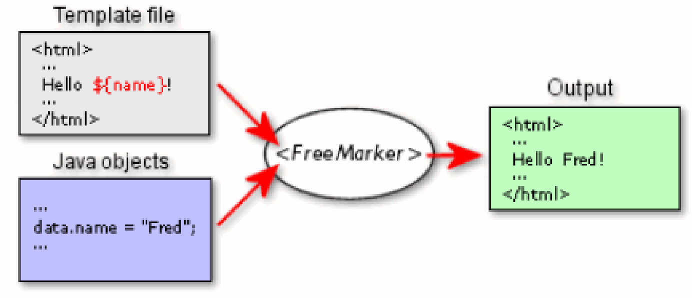


Figura 24. Freemarker

Para realizar una transformación usando una instancia de freemarker hay q:

1. Establecer el ClassPathConfiguration
2. Indicar el modelo de datos(diccionario) sobre el q se aplicará a la plantilla (sus variables se resolverán en función del diccionario)
3. indicar la plantilla a aplicar
4. indicar el lugar donde se almacenará el resultado generado

Con el objetivo de centralizar todos los accesos a Freemarker, se ha creado la clase TemplateEngine. En el constructor de esta clase se realizan los puntos 1-3 descritos anteriormente, y proporciona un método en el que se realiza el punto 4 y se ejecuta la transformación.

```java
public TemplateEngine(String templateName, HashMap data) throws
IOException, TemplateException{
	this.data=data;
	this.templateName=templateName;
	ClassPathConfiguration classPathConfiguration= new
ClassPathConfiguration(Activator.class,"");
	Configuration cfg =
classPathConfiguration.getConfiguration();
	tempServiceInterface = cfg.getTemplate(templateName);
	}

public void run(Writer out ) throws TemplateException,
IOException{
		tempServiceInterface.process(data, out);
		out.flush();
		System.out.print("WebPDA: Fin generacion");
	}
}
```

En el bundle desarrollado para la interfaz web PDA registra un servlet en el servidor web. Este servlet atiende las peticiones obteniendo la información necesaria del sistema pervasivo por medio del Controller y aplicándole a ésta una plantilla de freemarker.

Para registrar el servlet en el método Stara del Activator, debemos primero buscar el servicio httpService, y una vez encontrado registrar el servlet, en la dirección indicada, y los recursos gráficos (imágenes y hoja de estilo) que va a utilizar la web. En el caso en el que no se pueda encontrar el servicio httpService se suscribirá la clase Activator para ser notificada de registros de nuevos servicios:

```java
private void getHttpService(BundleContext context){
		if (this.httpTracker == null ){
			this.httpTracker = new
		ServiceTracker(context,HttpService.class.getName(),
		null);
			this.httpTracker.open();
		}
		Object [] services = this.httpTracker.getServices();
		if (!(services == null))
		{
			if (services[0].getClass().getName().equals(
	"com.prosyst.mbs.impl.framework.ServiceReferenceImpl"))
			this.httpService = (HttpService)
			this.context.getService((ServiceReference)
			services[0]);
		else
			this.httpService = (HttpService) services[0];
		}
	}
```

```
private void registerWebUI(BundleContext context, HttpService
httpService) throws Exception{


    WebUIPDA webUIPDA= new WebUIPDA(this.context,webUIPath);
    httpService.registerServlet(webUIPath, webUIPDA,null,sc);

      }
```

```
private void registerResources() {

        if (httpService!=null) {
            try {


      httpService.registerResources(webUIPath+"/resources","org/perv
ml/UI/WebUI/resources",sc);

            }
            catch(Exception ex) {
                System.out.println("register resource:
"+ex.getMessage());
            }
        }
    }
```

En segundo lugar debemos sobrescribir los métodos doGet y doPost, de la clase HttpServlet de la que debe extender el servlet implementado, que se ejecutan en respuesta a peticiones de páginas web por parte del usuario al Servlet. En estos dos métodos se invoca al método implementado processRequest, que realiza dos tareas:

1. Determinar cual ha sido la acción realizada por el usuario. Las acciones pueden ser: listar los tipos de servicio, listar las localizaciones, buscar un servicio, mostrar la información relativa a un servicio (parte estática, parte dinámica, operaciones permitidas en el estado actual), o invocar una operación. Para distinguir cual es la opción del usuario usaremos parámetros que se le pasaran al servlet.

2. Generar la pagina HTML en respuesta a la acción del usuario:
    a. Para las tres primeras acciones se debe mostrar la página HTML correspondiente. Para ello, los pasos genéricos que se siguen son:
        i. Obtener la información necesaria del sistema pervasivo a través del Controller. En algunos casos puede ser necesario realizar un preproceso de dicha información.
        ii. Encapsular la información obtenida en un diccionario
        iii. Crear una instancia del motor de plantillas pasándole la plantilla a aplicar y el diccionario con los datos obtenidos

iv. Invocar la transformación especificando donde se escribirá el resultado

b. Para la acción de invocar una operación se siguen los siguientes pasos:
   i. Averiguar si el servicio se encuentra en un estado en el cual es posible ejecutar la operación. Sino lo está, se generará una página de error.
   ii. Recuperar los parámetros de la operación, si los tiene.
   iii. Invocar la operación sobre el Controller, indicando el componente, la operación seleccionada y un array con los parámetros de dicha operación.
   iv. Realizar los pasos del punto 2 a para reflejar el nuevo estado del componente después de haber realizado la operación.

```
private void generateKindsList() throws TemplateException,
IOException{

      if (!searchingValue.equals("SPECIFICKIND")){
            String [] kinds = controller.getKindsOfservice();
            HashMap kindsMap= new HashMap();
            kindsMap.put("kinds",kinds);
            kindsMap.put("path",serviceWebPath);
            kindsMap.put("searchingValue",searchingValue);
           TemplateEngine templateEngine = new
                   TemplateEngine("WebUIKindsPDA.ftl", kindsMap);
            templateEngine.run(out);
            }
      else{
this.generateServiceList(controller.getServicesByKindOfService(
                   this.specific), "Kind: ", specific, "KIND");
}
```

```
private void generateServiceList(String [] servicePIDs, String
listType, String backLinkName, String type) throws IOException,
TemplateException{
      HashMap servicesMap= new HashMap();
      Service [] services= new Service [servicePIDs.length];
      for (int i=0; i<services.length;i++){
             services[i]=new Service(servicePIDs[i],
controller.getAliasOfService(servicePIDs[i]));
      }
      if (type.equals("LOCATION"))
             this.specific=getPreviousLocation(specific);
           String backURL= this.request.getRequestURI() +
"?searchingType="+type +"&searchingValue=" + this.specific;
           servicesMap.put("services",services);
           servicesMap.put("listType",listType);
           servicesMap.put("backLinkName",backLinkName);
           servicesMap.put("backURL",backURL);
           servicesMap.put("path",serviceWebPath);
           TemplateEngine templateEngine = new
TemplateEngine("WebUIServicesListPDA.ftl", servicesMap);
           templateEngine.run(out);
 }
```

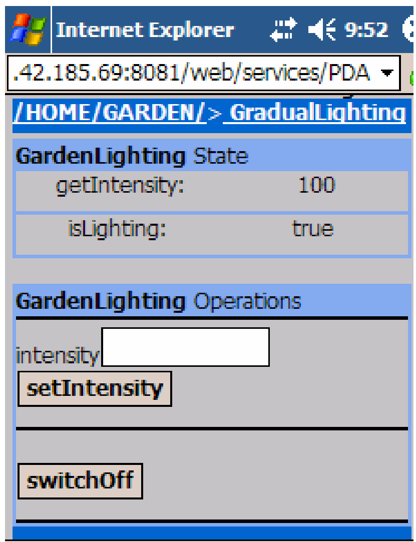


Figura 25. Interfaz para PDA del componente GardenLighting

10.2. Interfaz Web

Para la web de escritorio se sigue la misma estrategia aplicada a la web de PDA. Se utiliza el motor de plantillas Freemarker para aplicar plantillas a datos extraídos del sistema pervasivo utilizando el Controller.

Pese a que la información que se muestra al usuario en ambos casos es la misma, la disposición de esta en las páginas web es diferente. Esto es debido a las diferentes características de visualización entre un dispositivo móvil y un ordenador de sobremesa, como por ejemplo, tamaño de pantalla y relación de aspecto.

Figura 26. Interfaz Web del componente GardenLighting

11. Conclusiones

En este proyecto se ha presentado el desarrollo completo de un sistema pervasivo, de dimensiones muy cercanas a la realidad, aplicando un método de producción de software dirigido por modelos. Además, se ha presentado la implementación de un framework que aumenta el nivel de abstracción de la tecnología OSGi proporcionando constructores similares a los que define PervML.

Para construir el sistema pervasivo:1) se ha descrito el sistema mediante el lenguaje PervML diseñado para la especificación de sistemas pervasivos, 2) se ha generado código Java mediante un compilador de modelos, 3) se han desarrollado los drivers encargados de interactuar con los dispositivos o sistemas software que finalmente proporcionan la funcionalidad a los usuarios, realizando, además, su correspondiente instalación física, y 4) se han desarrollado diferentes interfaces para el control de los servicios proporcionados por el sistema pervasivo.

Los drivers desarrollados en este proyecto son reutilizables para futuros casos de estudio, si se utilizan el mismo tipo de dispositivo, simplemente cambiando sus identificadores.

También las interfaces son reutilizables para cualquier caso de estudio, ya que son independientes de los servicios que se ofrezcan, puesto que son capaces de conocer en tiempo de ejecución los servicios existentes en el sistema y la funcionalidad que aportan.

Por último concluir, que al aplicar un método dirigido por modelos, la descripción de la funcionalidad del sistema resulta independiente de los dispositivos seleccionados para implementarla. Además, toda la especificación es independiente de detalles dependientes de tecnología o fabricante, ya que estos se encuentran encapsulados en la capa de drivers. Gracias a esto, es posible cambiar de tecnología y/o de fabricante remplazando los drivers, de modo que la funcionalidad proporcionada por el sistema continúa siendo la misma.

11.1. Trabajos futuros

Se pretende extender la expresividad de PervML y las características del framework de implementación para (1) permitir la especificación de interfaces de usuario más ricas, quizá aplicando el enfoque propuesto por las SmartTemplates [13] y/o utilizando un lenguaje de especificación de interfaces de usuario como UsiXML [14]; y (2) para permitir la especificación de manera explícita de características sensibles al contexto.

También se pretende desarrollar nuevos drivers para controlar otros dispositivos (sensores de humo, aspersores, etc.), así como drivers software (envío de correo, mensajería instantánea, etc.) que ofrezcan nuevas funcionalidades.

11.2. Publicaciones

Como ya se ha comentado en el presente documento, el Proyecto Final de Carrera se ha desarrollado en el contexto del grupo de investigación OO-Method, lo que ha permitido la participación en varias publicaciones de carácter científico. Enumeramos a continuación dichas publicaciones:

- "**Applying a Model-Driven Method to the Development of a Pervasive Meeting Room**", *Javier Muñoz, Estefanía Serral, Carlos Cetina and Vicente Pelechan,* ERCIM News, April 2006, vol. 65, pp. 44-45, ISSN: 0926-4981

- "**Aplicación del Desarrollo Dirigido por Modelos a los Sistemas Pervasivos: Un Caso de Estudio**", *Javier Muñoz, Vicente Pelechano, Estefanía Serral,* II Congreso IberoAmericano sobre Computación Ubicua (CICU 2006), Alcalá de Henares (Spain), 7-9 June 2006, pp. 171-178, ISBN: 84-8138-703-7

- "**Un Framework basado en OSGi para el Desarrollo de Sistemas Pervasivos**", *Javier Muñoz, Carlos Cetina, Estefanía Serral, Vicente Pelechado,* 9 Workshop Iberoamericano de Ingenieria de Requisitos y Ambientes Software (IDEAS2006), La Plata (Argentina), 24 - 28, Apr 2006 pp. 257 – 270

- "**Providing platforms for developing pervasive systems with MDA. An OSGi metamodel**", Javier Muñoz, Vicente Pelechano, Estefanía Serral, X Jornadas de Ingeniería de Software y Base de Datos (JISBD), Granada (Spain), September 2005, pp. 19 - 26, ISBN: 84-9732-434-X

12. Referencias

[1] EIBA. http://www.eiba-es.com/dept.asp?deptid=3, 3-05-06

[2] OSGI. Home page. http://www.osgi.org, 23-02-06

[3] Applying a Model-Driven Method to the Development of a Pervasive Meeting Room
Javier Muñoz, Estefanía Serral, Carlos Cetina and Vicente Pelechano
ERCIM News
April 2006
vol. 65, pp. 44-45, ISSN: 0926-4981

[4] Frank Buschmann, Regine Meunier, Hans Rohnert, Peter Sommerlad, and Michael Stal. Pattern-Oriented Software Architecture, volume 1: A System of Patterns. Wiley, 1996.

[5] Aplicación del Desarrollo Dirigido por Modelos a los Sistemas Pervasivos: Un Caso de Estudio
Javier Muñoz, Vicente Pelechano, Estefanía Serral
II Congreso IberoAmericano sobre Computación Ubicua (CICU 2006), Alcalá de Henares (Spain)
7-9 June 2006
pp. 171-178, ISBN: 84-8138-703-7

[6] Pelechano, V., "Managing Taxonomic Relationships in Automatic Software Production Enviroments: A Pattern based Approach". UMI ProQuest Digital Dissertationns. 2001.]

[7] "Implementing UML Association, Aggregation and Composition. A Particular Interpretation based on a multidimensional Framework.", Albert, M., Pelechano, V., Fons, J., Ruiz, M., Pastor, O., 15th Conference On Advanced Information Systems Engineering (CAISE2003). 2003.].

[8] Object Management Group.
Model Driven Architecture Guide, 2003.

[9] Jack Green_eld, Keith Short, Steve Cook, and Stuart Kent.
Software Factories. Wiley Publising Inc., 2004.

[10] "Un Framework basado en OSGi para el Desarrollo de Sistemas Pervasivos". Javier Muñoz, Carlos Cetina, Estefanía Serral, and Vicente Pelechano. In 9 Workshop Iberoamericano de Ingeniería de Requisitos y Ambientes Software (IDEAS 2006), La Plata (Argentina), April 2006.

[11] Prosyst. Home page.http://www.prosyst.com/, 7-03-06

[12] "Un framework para la simulación de sistemas pervasivos"
Javier Muñoz, Idoia Ruiz, Vicente Pelechano, Carlos Cetina
Simposio sobre Computación Ubicua e Inteligencia Ambiental (UCAmI'05), Granada (Spain)
September 2005
pp. 181 - 190, ISBN: 84-9732-442-0

[13] Jeffrey Nichols, Brad A. Myers, and Kevin Litwack. Improving Automatic Interface Generation with Smart Templates. In Intelligent User Interfaces (IUI) 2004, pages 286_288, Funchal, Portugal, January 2004.

[14]Quentin Limbourg and Jean Vanderdonckt. Engineering Advanced Web Applications, chapter Usixml: A User Interface Description Language Supporting Multiple