UNIVERSIDAD POLITECNICA DE VALENCIA

Proyecto Final de Carrera

Ingeniero Técnico en Informática de Gestión

Gestión domótica del hogar mediante la tecnología OSGi

Abril 2011, Valencia

Jesús Martín Sayas
jesmarsa@ei.upv.es

Dirigido por:

Joan Fons i Cors

jjfons@dsic.upv.es

Nacho Mansanet Benavent

imansanet@dsic.upv.es

INDICE

INDICE DE ILUSTRACIONES

1. INTRODUCCIÓN

En la actualidad existe mucha tecnología implicada en acciones cotidianas. Hoy en día se puede hacer prácticamente de todo a tiempo real y en cualquier lugar, no hay más que observar la domótica, que pretende que se pueda manejar una casa real mediante dispositivos tecnológicos. Así, y mediante esta idea, podemos manejar por ejemplo la iluminación de toda la casa vía Internet o permitir que la calefacción se encienda y se apague automáticamente en el instante que queramos, entre otros.

Pero ¿que motiva a la gente a que se use la domótica? ¿Qué finalidad buscamos a través de la domótica que no pueda ofrecernos otra tecnología? Dentro de las aplicaciones principales de la domótica, encontramos varias que hacen que nos decantemos por esta tecnología para controlar y automatizar lo que denominaremos un hogar digital.

Ahorro Energético

Una de las principales ventajas del uso de la domótica es el ahorro energético. A pesar de que no sea un aspecto que sea tangible al usuario, es uno de los más importantes, puesto que hoy en día uno de los problemas comunes a todos los seres humanos es la cantidad de energía que se derrocha para vivir en un hogar día a día.

Para conseguir este ahorro energético no simplemente basta con sustituir los aparatos o sistemas del hogar por otros que consuman menos sino que también se trata de hacer una gestión eficiente de los mismos, en concreto:

- Climatización: programación y zonificación. Se puede programar la climatización del hogar según determinadas zonas (para que no se aireé todo el hogar por completo, sino sólo la zona que deseemos) y/o determinar en qué momentos del día se debe utilizar.
- Gestión eléctrica:
    - Racionalización de cargas eléctricas: desconexión de equipos de uso no prioritario en función del consumo eléctrico en un momento dado.
    - Gestión de tarifas, derivando el funcionamiento de algunos aparatos a horas de tarifa reducida.
- Uso de energías renovables.

Confort

Otra de las ventajas, no menos importante, es el confort. El uso de la domótica en un hogar digital proporciona adicionalmente un confort en el día a día que facilita que las acciones que se llevan a cabo sean más fáciles. Dichas acciones pueden ser de carácter tanto pasivo, como activo o mixtas, dependiendo de si el usuario interviene en ellas o no:

- Iluminación:
    - Apagado general de todas las luces de la vivienda
    - Automatización del apagado / encendido en cada punto de luz.

    - Regulación de la iluminación según el nivel de luminosidad ambiente.
- Automatización de todos los distintos sistemas / instalaciones / equipos dotándolos de control eficiente y de fácil manejo.
- Integración del portero al teléfono, o del video portero al televisor.
- Control vía Internet.
- Gestión Multimedia y del ocio electrónicos
- Generación de macros y programas de forma sencilla para el usuario.

Seguridad

La seguridad que proporcionan estas tecnologías es bastante importante. La finalidad que se consigue es proteger tanto los bienes patrimoniales (cualquier elemento de la casa) como la seguridad personal. Esto se consigue gracias a varios elementos:

- Alarmas de intrusión (anti intrusión). Se utilizan para detectar o prevenir la presencia de personas extrañas en una vivienda o edificio, en concreto:
    - Detección de un posible intruso (Detectores volumétricos o perimetrales)
    - Cierre de persianas puntual y seguro
    - Simulación de presencia

- Alarmas de detección de incendios, fugas de gas, escapes de agua, concentración de monóxido en garajes cuando se usan vehículos de combustión.
- Alerta médica. Tele asistencia.
- Acceso a Cámaras IP

Comunicaciones

Una de las mayores ventajas que detecta el usuario al utilizar un hogar digital es la cantidad de dispositivos, sistemas o infraestructuras de comunicaciones que posee el hogar. Esto permite, en primer plano, una facilidad de uso y una diversidad de usuarios distintos que pueden utilizar el hogar digital.

Entre estos distintos sistemas / dispositivos / infraestructuras encontramos:

- Ubicuidad en el control tanto externo como interno, control remoto desde Internet, PC, mandos inalámbricos (p. ej. PDA con WiFi), entre otros.
- Tele asistencia
- Tele mantenimiento
- Informes de consumo y costes
- Transmisión de alarmas
- Intercomunicaciones

Accesibilidad

Bajo este epígrafe se incluyen las aplicaciones o instalaciones de control remoto del entorno que favorecen la autonomía personal de personas con limitaciones funcionales, o discapacidad. El concepto "diseño" para todos es un movimiento que pretende crear la sensibilidad necesaria para que al diseñar un producto o servicio se tengan en cuenta las necesidades de todos los posibles usuarios, incluyendo las personas con diferentes capacidades o discapacidades, es decir, favorecer un diseño accesible para la diversidad humana. La inclusión social y la igualdad son términos o conceptos más generalistas y filosóficos. La domótica aplicada a favorecer la accesibilidad es un reto ético y creativo pero sobre todo es la aplicación de la tecnología en el campo más necesario, para suplir limitaciones funcionales de las personas. El objetivo no es que las personas con discapacidad puedan acceder a estas tecnologías, porque las tecnologías en si no son un objetivo, sino un medio. El objetivo de estas tecnologías es favorecer la autonomía personal. Los destinatarios de estas tecnologías son todas las personas, ya que por enfermedad o envejecimiento, todos somos o seremos discapacitados, más pronto o más tarde.

2. CONTEXTO TECNOLÓGICO Y PROBLEMÁTICA. IDEAS DE SOLUCIÓN.

La problemática que nos encontramos ante este proyecto es bien simple. Disponemos de dispositivos domóticos en un hogar digital, y necesitamos disponer de un software que soporte todos estos dispositivos y no solo eso, sino que estos dispositivos interactúen entre sí y hagan la vida del usuario más simple y más cómoda (entre otras ventajas).

Para ello hacemos uso de una tecnología que, a pesar de no ser la única que solvente el problema, es la que más ventajas nos ofrece y la más cómoda a la hora de trabajar con ella: OSGi.

Esta tecnología permite que elementos domóticos como pueden ser cualquier elemento de un hogar (ventanas, iluminación, puertas…) puedan interaccionar entre sí y, lo que es más importante, puedan interaccionar con el usuario. Esto se permite gracias a unos drivers (que han sido desarrollados en el presente proyecto) y que enlazan el dispositivo físico con los llamados Binding Providers para así dar servicio a las necesidades del usuario. Estos Binding Providers son utilizados por los servicios para completar su funcionalidad.

Así podremos disponer de dispositivos físicos, como una bombilla, un interruptor, una ventana, un televisor, etc; y servicios como por ejemplo iluminación, reproducción multimedia, ventilación, detección de humos, detección de presencia, etc. Estos servicios se pueden comunicar entre sí, para conseguir los efectos esperados por el usuario, como que por ejemplo al detectar la presencia se encienda la luz del habitáculo, que cuando se

encienda el aire acondicionado se cierren las ventanas, o que cuando se enciendan las luces de un determinado sitio suene el hilo musical en esa zona concreta. Así se consigue un ahorro significativo de la energía consumida por todos estos dispositivos, como una comodidad añadida a la hora de realizar las tareas cotidianas por el usuario.

El mismo usuario puede definir unos ámbitos de uso, como pueden ser distintos tipos de iluminación y registrarlos en el sistema, así puede definirse iluminación nocturna o diurna, en la cual se enciendan las luces y se bajen las persianas, y viceversa, o ambiente de verano / invierno, donde se active o se desactive la ventilación o la calefacción dependiendo de la estación en la que nos encontremos.

Bajo esta tecnología, nos basamos en una arquitectura existente, llamada Calimero, que nos ofrece una serie de clases Java que nos permiten representar los dispositivos como tales e interactuar con ellos. Esta arquitectura es de Open Source, y esto nos permite más facilidades a la hora de mantenerla. Para poder representar todo esto, Calimero nos ofrece lo que se llama un túnel, que conecta los dispositivos físicos con el PC y nos permite comunicarnos con ellos.

Más adelante explicaremos detalladamente la tecnología OSGi y la arquitectura de Calimero, puesto que se permiten un análisis más pormenorizado.

3. TECNOLOGÍA RELACIONADA. OSGI

OSGi (Open Services Gateway Initiative) es una alianza que fue fundada en marzo de 1999 con la finalidad de desarrollar estándares para el desarrollo de pasarelas de servicios (entre ellos el estándar de OSGi). Entre los miembros de dicha alianza hay nombres de empresas conocidas y de diferentes áreas de negocio, como pueden ser Motorola, Nokia, Siemens, Telefónica y Oracle entre otros muchos (más de 35).

Este consorcio está dividido en diferentes grupos de trabajo, con distintos objetivos dentro de la especificación:

- Grupos de expertos:
    - Core Platform (CPEG): Define las especificaciones básicas de la plataforma. El CPEG se centra en los componentes que forman el entorno de ejecución OSGi y los servicios fundamentales para todos los entornos de estas características.
    - Mobile (MEG): Define los requisitos y especificaciones orientados a la plataforma de servicios OSGi para dispositivos móviles.
    - Vehicle (VEG): Se centra en la adaptación y ampliación de la plataforma destinada a los entornos empotrados en vehículos.
    - Entreprise (EEG): Este grupo es bastante reciente, y está orientado a definir requisitos y especificaciones técnicas para adaptar y ampliar la plataforma de servicios OSGi al ámbito del software empresarial.

    - Residential (REG): Define los requisitos y especificaciones para adaptar y ampliar la plataforma de servicios OSGi al ámbito de las plataformas residenciales.
- Grupos de trabajo de marketing:

  Fueron creados para ayudar y promover la Alianza OSGi, mediante una serie de estrategias de publicidad.

- Grupos de trabajo de requerimientos de mercado:

  Como su propio nombre indica, se trata de grupos de trabajo que crean documentos de requisitos, los cuales describen las principales exigencias del mercado.

La especificación de OSGi define un sistema de módulos y servicios, bajo la tecnología Java, que implementan un modelo de componentes dinámicos y completo, algo que hasta la fecha era impensable. Es decir, OSGi define una arquitectura software mínima y necesaria donde las aplicaciones pueden descubrir y utilizar otros servicios de forma dinámica, sin necesidad de reiniciar todo el sistema. Así, por ejemplo, se puede tener una instalación domótica con un interruptor y una luz, y añadir y utilizar otro nuevo interruptor inmediatamente, sin necesidad de tener que avisar al sistema de que hay un nuevo dispositivo doméstico.

FRAMEWORK

Uno de los componentes básicos de OSGi es el *framework, que* ofrece un entorno estándar para las aplicaciones.

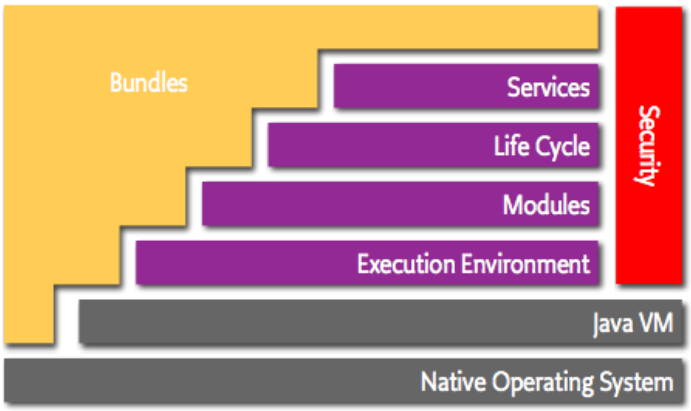

Ilustración 1 - Framework de OSGi

La parte principal del framework son los llamados *bundles* que engloban las cuatro características típicas: son *módulos* que tienen un *ciclo de vida*, entrelazados entre sí mediante *servicios* dentro de un *entorno de ejecución*.

La funcionalidad del framework se divide en las siguientes capas:

- Seguridad: Capa de seguridad que está profundamente entrelazada a lo largo de todas las capas y que está basada en el modelo de seguridad de Java 2. Este modelo proporciona un modelo global para comprobar el código de acceso a los recursos.
- Módulos: Esta capa define el modelo de modularización, que define a su vez las reglas para el intercambio de paquetes java entre los Bundles. Es decir, esta capa se encarga de gestionar la interconexión entre Bundles.

- Bundles.
- Gestión de servicios.
- Ciclos de vida.

A las capas de bundles, gestión de servicios y ciclos de vida les dedicaremos capítulos aparte a partir de ahora, puesto que son más extensas y dedican un mayor trato.

CICLO DE VIDA

Esta capa dentro del framework es una de las más importantes, puesto que define una serie de estados por los cuales puede pasar cualquier bundle, y que definen la vida del mismo. Estos estados son: **instalación, desinstalación, arranque, parada y actualización**. El ciclo de vida, así mismo, también permite resolver las dependencias de los módulos de forma dinámica.

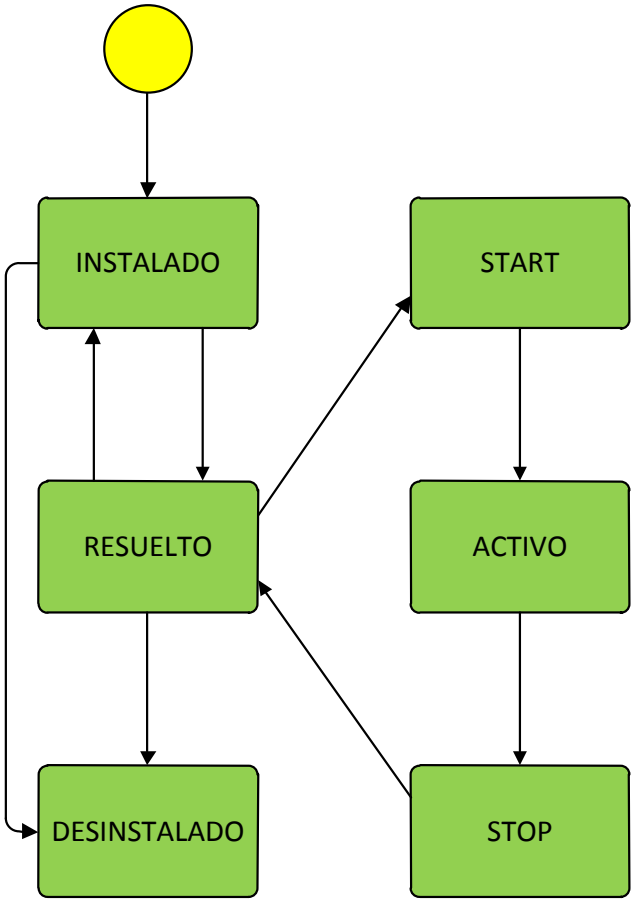

Ilustración 2 – Ciclo de vida elemental

BUNDLES

Los bundles son el mecanismo utilizado para distribuir e instalar aplicaciones y servicios en la plataforma.

Un bundle es un Archivo Java (JAR) con la parte mínima de una aplicación OSGi, además, tienen un ciclo de vida restringido que es administrado externamente por el framework. El módulo de registro de servicios provee el mecanismo de manejo de los mismos, por eso, un bundle está diseñado para ser un conjunto de servicios que cooperan en el registro de servicios. Además, un bundle puede llevar información descriptiva sobre sí mismo en un archivo llamado manifest. El archivo manifest, normalmente está ubicado en la carpeta META-INF y su nombre debe ser MANIFEST.MF. Este archivo contiene varias etiquetas entre las que cabe destacar las siguientes:

- Bundle-Activator: ruta donde se encuentra la clase activator.
- Bundle-Name: nombre del servicio.
- Bundle-Description: descripción del bundle.
- Import-Package: paquetes que necesita para ejecutarse, normalmente se importa: org.osgi.framework.

Estas son imprescindibles y debe respetarse el orden en el que se han presentado anteriormente para conseguir que el bundle funcione correctamente. Además, existen otras etiquetas opcionales como pueden ser:

- Bundle-vendor: indica el nombre del autor del bundle.
- Bundle-version: identifica la versión del bundle.

GESTIÓN DE SERVICIOS

En concreto, un servicio OSGi se define mediante una interfaz, especificando los métodos públicos e implementándose como un objeto. El bundle es el encargado de registrar el servicio en el Servicio de Registros y así, su funcionalidad estará disponible para otros bundles. En general, los servicios registrados son referenciados mediante el Servicio de Referencia de Objetos para mantener las propiedades y otra meta-información sobre el servicio. En la siguiente figura, puede observarse un ejemplo de cómo registrar un servicio y obtenerlo:

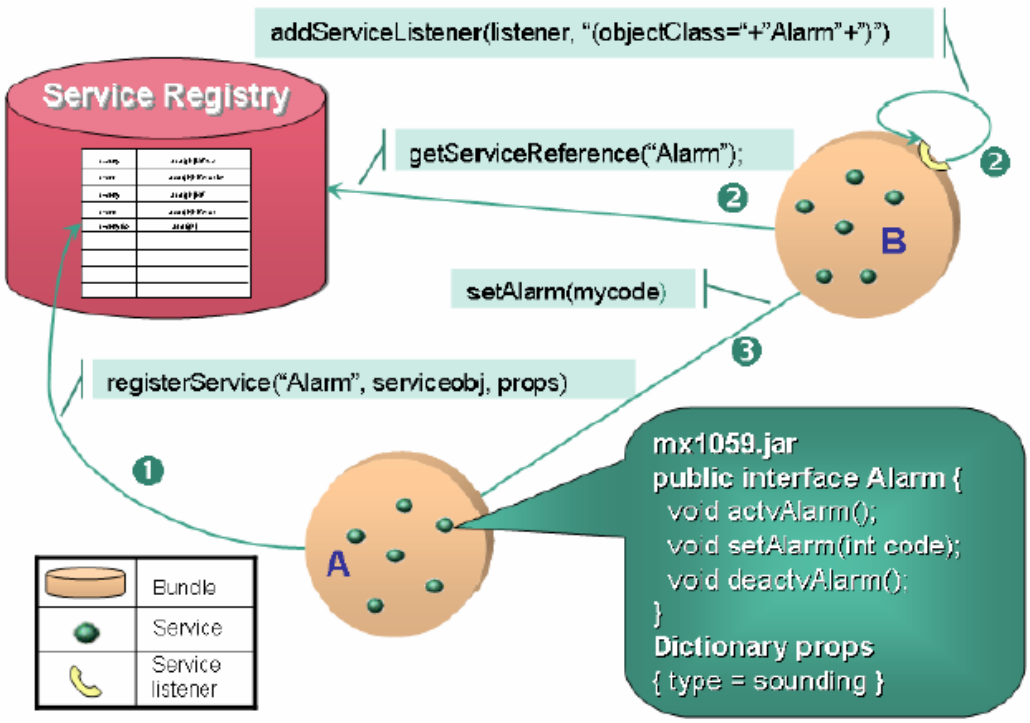

Ilustración 3 – Operaciones con servicios

1. Registro de servicios.

   Cuando un método registra un servicio (invocando al método registerService ()) se especifica el nombre de la interfaz que lo implementa y también, es recomendable incluir una descripción utilizando una colección de pares: clave/valor. Por ejemplo, en la figura anterior, el bundle A registra un servicio (serviceobj) junto con un diccionario de propiedades (props). El registro sería:

   registerService ("Alarm",serviceobj,props).

2. Obtener servicios.

   Un bundle puede obtener un servicio OSGi mediante una búsqueda o un evento proporcionado por el framework (Service Listeners). Este último mecanismo, notifica a los bundles automáticamente cuando un servicio ha sido registrado, modificado o desinstalado además, pueden ser filtrados por el oyente. En el ejemplo anterior, el bundle B crea un oyente para obtener información sólo de los servicios de alarma:

   AddServiceListener (listener,"(objectClass="+"Alarm"+")").

   De todos modos, se pueden acceder a servicios específicos activando la búsqueda en el Registro de Servicios utilizando el método: getServiceReference () cuyo parámetro de entrada es el nombre de la interfaz que implementa el servicio. Siguiendo el ejemplo anterior, el bundle B puede obtener la referencia del objeto correspondiente al servicio registrado en el bundle A como sigue:

   ref=getServiceRefenrence("Alarm").

3. Utilizar servicios.

   Una vez que el bundle B ha obtenido la referencia del objeto, debe obtener el objeto del servicio para ser capaz de usar los métodos asociados al servicio. En este caso, para el servicio Alarm hay tres métodos disponibles: actvAlarm, setAlarm, deactvAlarm y para invocarlos sería:

   ```
   alarmSrv=getService(ref)
   alarmSrv.setAlarm(mycode)
   ```

   Por otro lado, el framework de OSGi ofrece la posibilidad de refinar la búsqueda utilizando el método getServiceReferences(), cuyos parámetros de entrada son:

   1) El nombre de la interfaz que implementa el servicio.

   2) Propiedades de búsqueda. Se especifican mediante un filtro cuya sintaxis está basada en el lenguaje LDAP (Lightweight Directory Access Protocol). Por ejemplo, para obtener un listado con las alarmas sonoras que pertenezcan a la interfaz Alarm, el bundle puede utilizar:

   ```
   Ref=getServiceReferences("Alarm","(type=sounding)")
   ```

   donde el filtro type=sounding es comparado con el diccionario de propiedades proporcionado en el registro.

4. ANÁLISIS DEL PROBLEMA. FUNCIONALIDAD Y EXTENSIONES.

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 fue 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. A continuación se describe tanto la estructura como la estrategia global de ejecución del framework.

ESTRUCTURA DEL FRAMEWORK

El framework se puede estructurar en dos partes bien diferenciadas: 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.

En este proyecto nos centraremos únicamente en la Capa Lógica, puesto que su desarrollo es uno de los objetivos principales del mismo.

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.

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 enlace (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 (Binding Providers), 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 cierren las ventanas de una sala cuando se active el aire acondicionado de la misma. Conceptualmente, los servicios de cierre de ventanas y activación del aire acondicionado no están relacionados entre sí pero, en un sistema en concreto, el usuario puede desear establecer una interacción entre ellos.

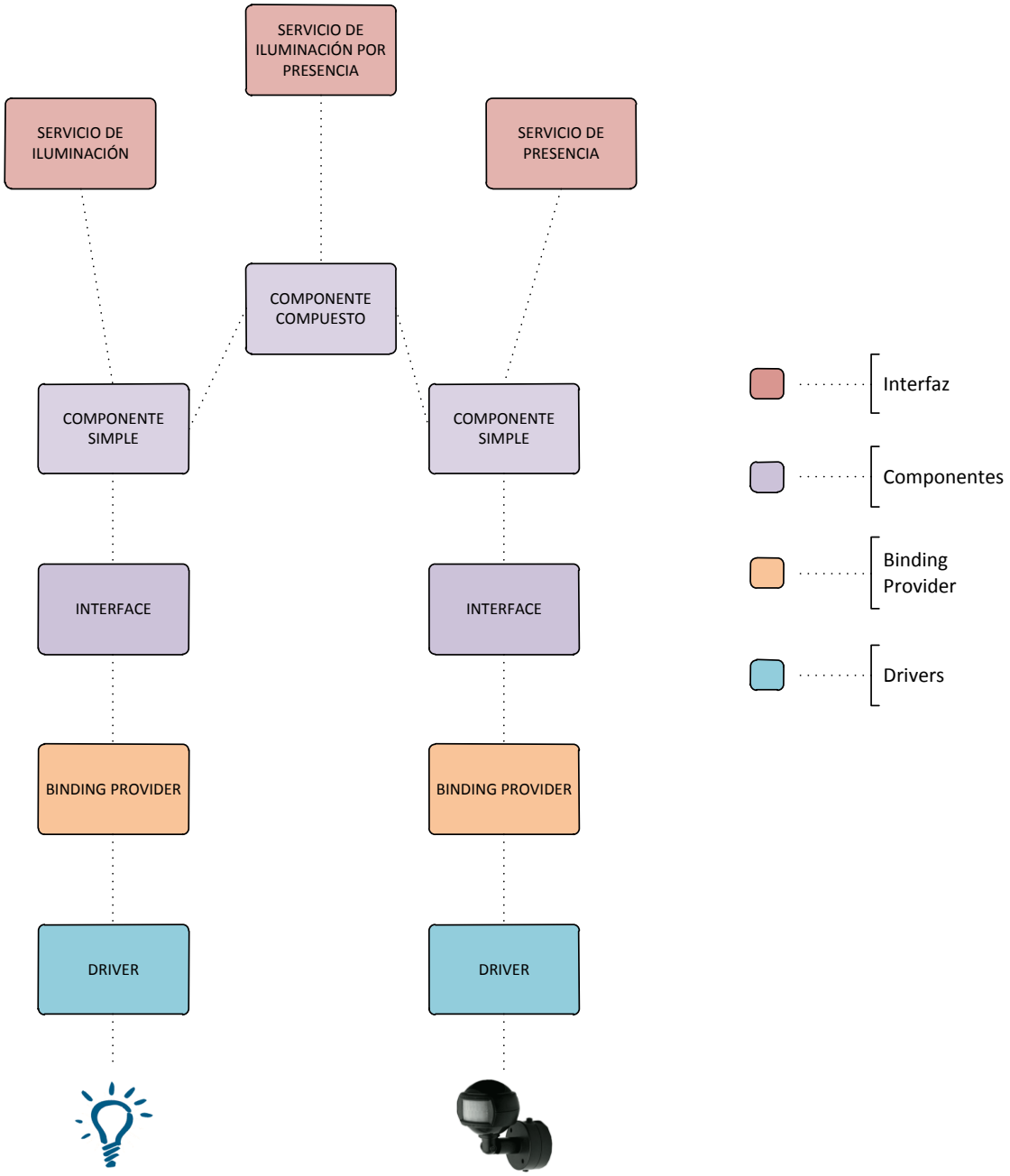

Ilustración 4 – Arquitectura básica de OSGi

En este ejemplo concreto, tenemos dos dispositivos, como son iluminación y sensor de presencia. Estos dos dispositivos están enlazados con sus proveedores de enlace (Binding Providers) a través de los drivers. Es decir, los proveedores de enlace hacen uso de los drivers para permitir completar

su funcionalidad. Un binding provider tiene dos funcionalidades principales: registrar el dispositivo en el sistema, permitiendo hacer referencia a él inequívocamente; y hacer uso de sus funcionalidades mediante los drivers.

En el siguiente nivel, las interfaces definen las llamadas básicas a estos Binding Providers , utilizando para ello los métodos que sean necesarios. Por ejemplo, en este caso, la interfaz de la iluminación contendrá las cabeceras de los métodos de encendido y apagado de la luz, mientras que el detector de presencia tendrá los métodos de encendido y apagado del detector de presencia.

En el siguiente nivel, los componentes simples lo único que hacen es definir el comportamiento de los métodos definidos en las interfaces. Así, el componente de iluminación dirá qué deben hacer los métodos de encendido y de apagado, mientras que el componente de detección de presencia se encargará de definir el comportamiento de detección de presencia.

A este nivel existen los componentes compuestos, que permiten que varios componentes interactúen entre sí. En este ejemplo tendría sentido, por ejemplo, que se encendiera la luz al detectar una presencia y viceversa. Este comportamiento se definiría en estos componentes compuestos.

ESTRATEGIA GLOBAL DE EJECUCIÓN

En este apartado se explicarán las reglas que definen las secuencias de acciones que suceden en el sistema cuando este se encuentra en funcionamiento, es decir, la estrategia que se decide seguir cuando el sistema domótico está en funcionamiento:

SECUENCIA DE ACCIONES

Existen dos posibles modos de iniciar una secuencia de acciones:

1. Un Driver notifica a un Binding Provider 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.

Cuando un Driver notifica a un BindingProvider que ha habido un cambio en el entorno, el Binding Provider 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 ilustración 5, 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 Binding Provider 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 Binding Providers 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.

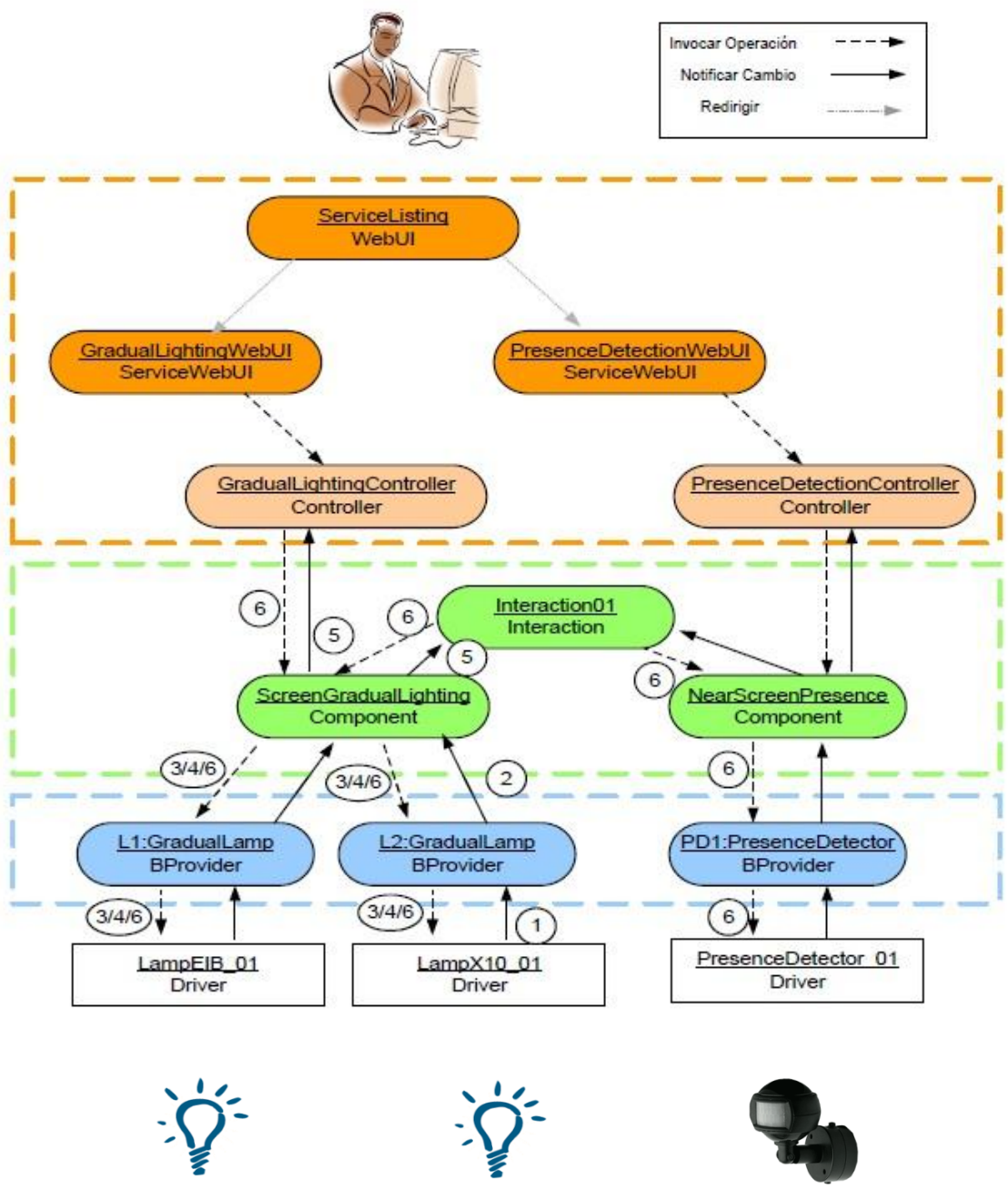

Ilustración 5 – Estrategia de ejecución en el caso de interfaz WEB

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 ilustración 6, 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.

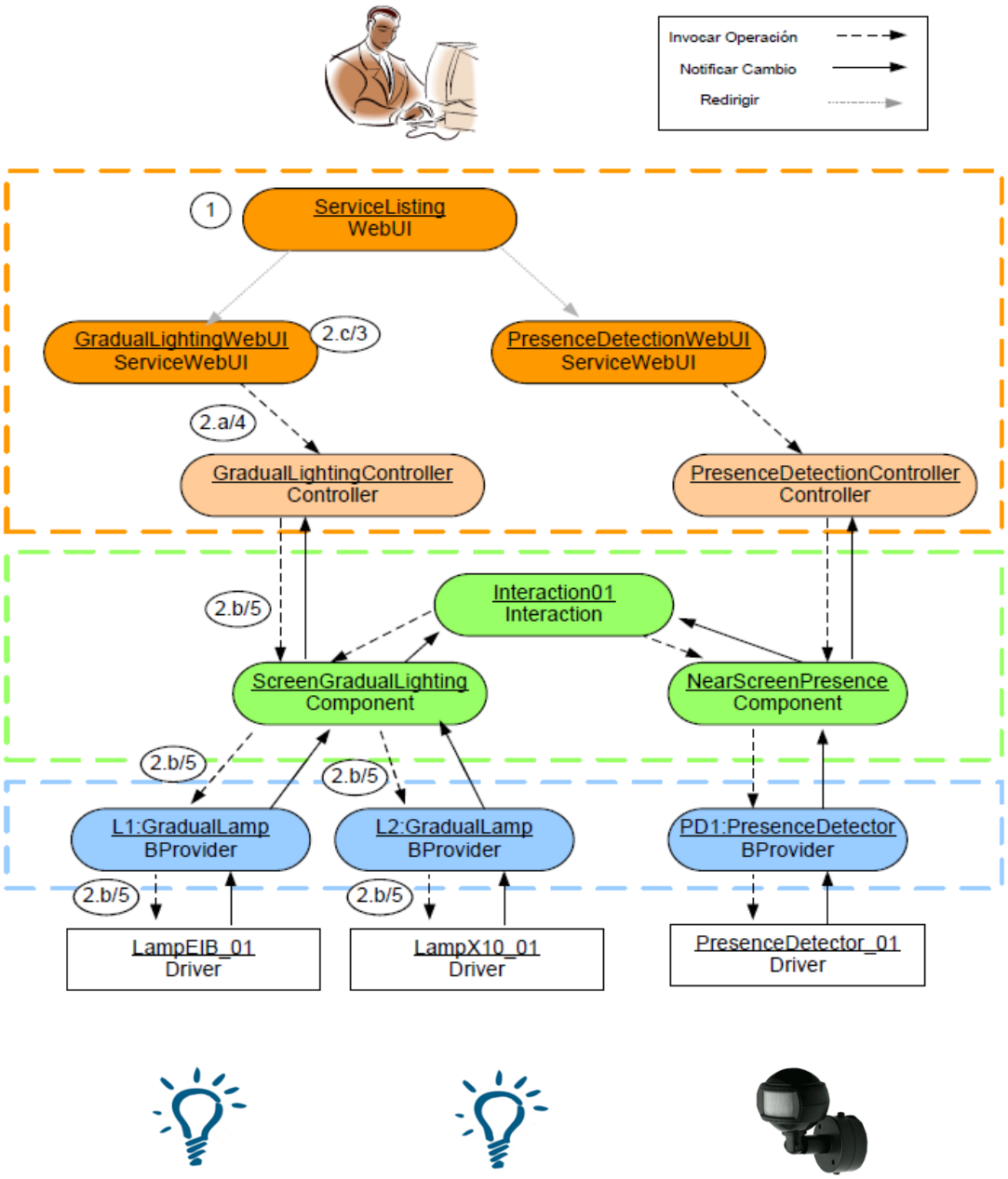

Ilustración 6 – Estrategia de ejecución en el caso de interfaz WEB

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 se 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 *Binding Providers*, 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 *Binding Providers* 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. DISEÑO DE LA SOLUCIÓN. DIAGRAMAS DE CLASE DE DISEÑO.

CALIMERO

¿QUÉ ES CALIMERO?

Para poder establecer una conexión a una red de dispositivos domóticos se utiliza lo que se conoce como Calimero. Se trata de un conjunto de API's desarrolladas en JAVA que facilitan este propósito además del envío como la recepción de datos mediante una conexión EIBNet/IP.

EIBnet/IP es un protocolo que permite el acceso a una instalación KNX/EIB mediante una conexión TCP/IP. Este protocolo sencillamente se basa en una serie de preguntas y respuestas HTTP que permiten establecer una comunicación entre los dispositivos.

DISEÑO ARQUITECTÓNICO

Uniendo las oportunidades que nos brindan tanto Calimero como EIBd (que se explicará en el siguiente punto), se ha desarrollado una API en java que permite el acceso a alto nivel a una instalación domótica.

A continuación se van a exponer las principales piezas que componen el túnel.

- ***DISPOSITIVO***

Se ha incorporado al sistema el concepto de dispositivo y se entiende como un aparato conectado a una instalación (*tunnel*) y que puede identificarse en esta mediante una dirección (*address*). Este además, va gestionar un tipo de dato (*DataType*) y tendrá un nombre único en el sistema (*Id*), que no en la

instalación. Esto quiere decir que en el sistema pueden convivir dos o más dispositivos con idéntica dirección y túnel, pero el *Id* debe ser único para cada dispositivo.

También indicaremos en el dispositivo un perfil que indicará si el dispositivo va a consumir datos (actuador) o a suministrarlos (sensor) al sistema.

Para ser más específicos, al dispositivo le indicaremos un tipo de los siguientes:

- Sensor
- Regulador Porcentual (0-100)
- Regulador Angular (0-360)
- Regulador Hexadecimal (0x00 – 0xFF)
- Dispositivos de movimiento vertical (Persianas: Arriba-Abajo-Parar)
- Dispositivos de tipo pulsador (Timbre)
- Dispositivos bi-estado (Interruptor)

Cabe indicar que para los sensores se ha proporcionado una extensión a esta interfaz que nos permite indicarle al sistema cuando deseamos recibir notificación de los valores: siempre que el dispositivo facilite un valor o solo si el valor ha cambiado desde el último que se nos ha notificado.

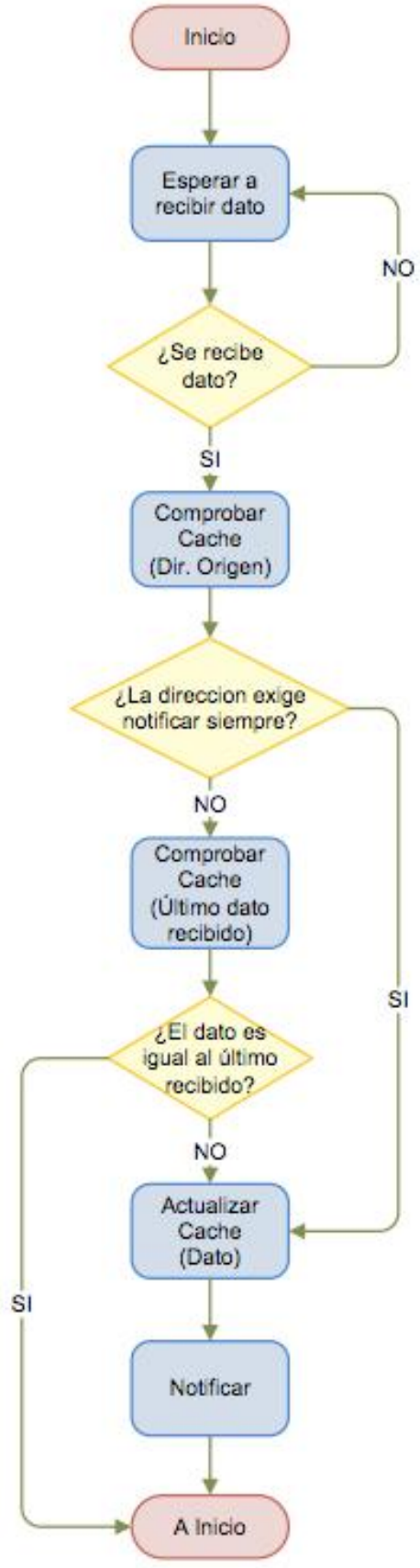

Ilustración 7 – Diagrama de flujo de los sensores

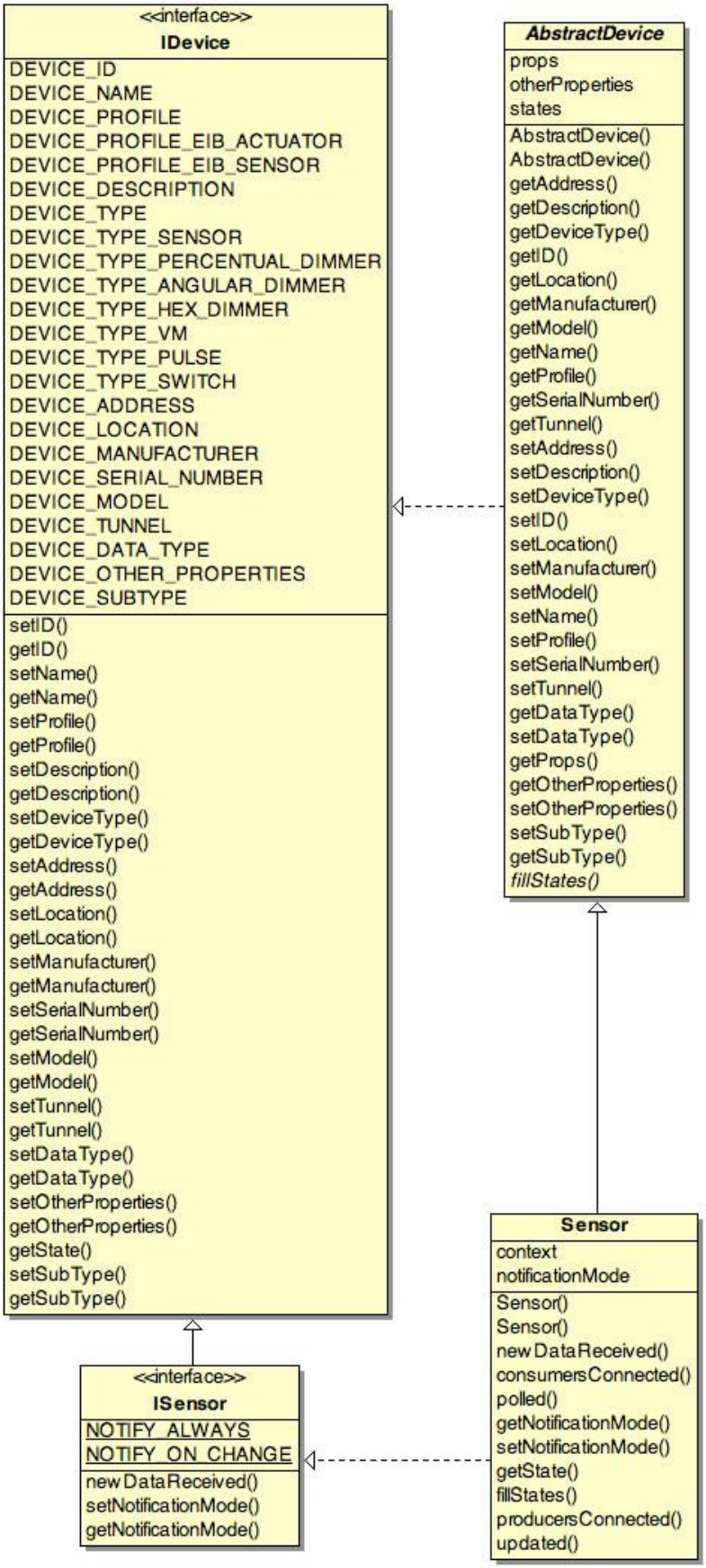

Ilustración 8 – Diagrama de clases de los dispositivos

- ***TÚNEL***

El concepto de túnel que manejamos en la arquitectura se refiere al de la pieza que actúa de intermediario entre los dispositivos físicos y su representación en componentes OSGi.

Todos los datos que vayan o vengan de la instalación física deberán pasar por el túnel adecuado que actuará a su vez de filtro.

Cada túnel se conectará con una y solo una instalación física a la que accederá vía EIBnet/IP comunicándose con un EIBd que estará conectado físicamente a la instalación mediante un cable USB. Además el túnel se identificará con un nombre que será único en el sistema.

Todo túnel presente en el sistema podrá actuar de 3 maneras respecto a los dispositivos físicos a que está vinculado:

- Solo puede enviar datos (solo controla los actuadores de la instalación).

  Se registrará solo bajo la interfaz *ITunnelWriter*

- Solo puede recibir datos (solo controla los sensores de la instalación).

  Se registrará solo bajo la interfaz *ITunnelReader*

- Puede tanto enviar como recibir datos.

Los túneles conservarán también una cache que conservará los datos que se envían y reciben del túnel para conservar el último dato. Por otra parte conservarán un histórico con todos los datos enviados y recibidos desde el túnel juntamente con el instante de tiempo en que ha ocurrido el evento. Contará también con una lista de actuadores y sensores asociados.

Cada túnel cuenta con métodos para escribir distintos tipos de valores (entero, booleano…) en los dispositivos.

Este componente actuará también a modo de factoría de drivers para dispositivos actuadores, es decir, se podrá solicitar a un túnel específico que facilite una implementación adecuada a ciertas propiedades para un dispositivo de este tipo.

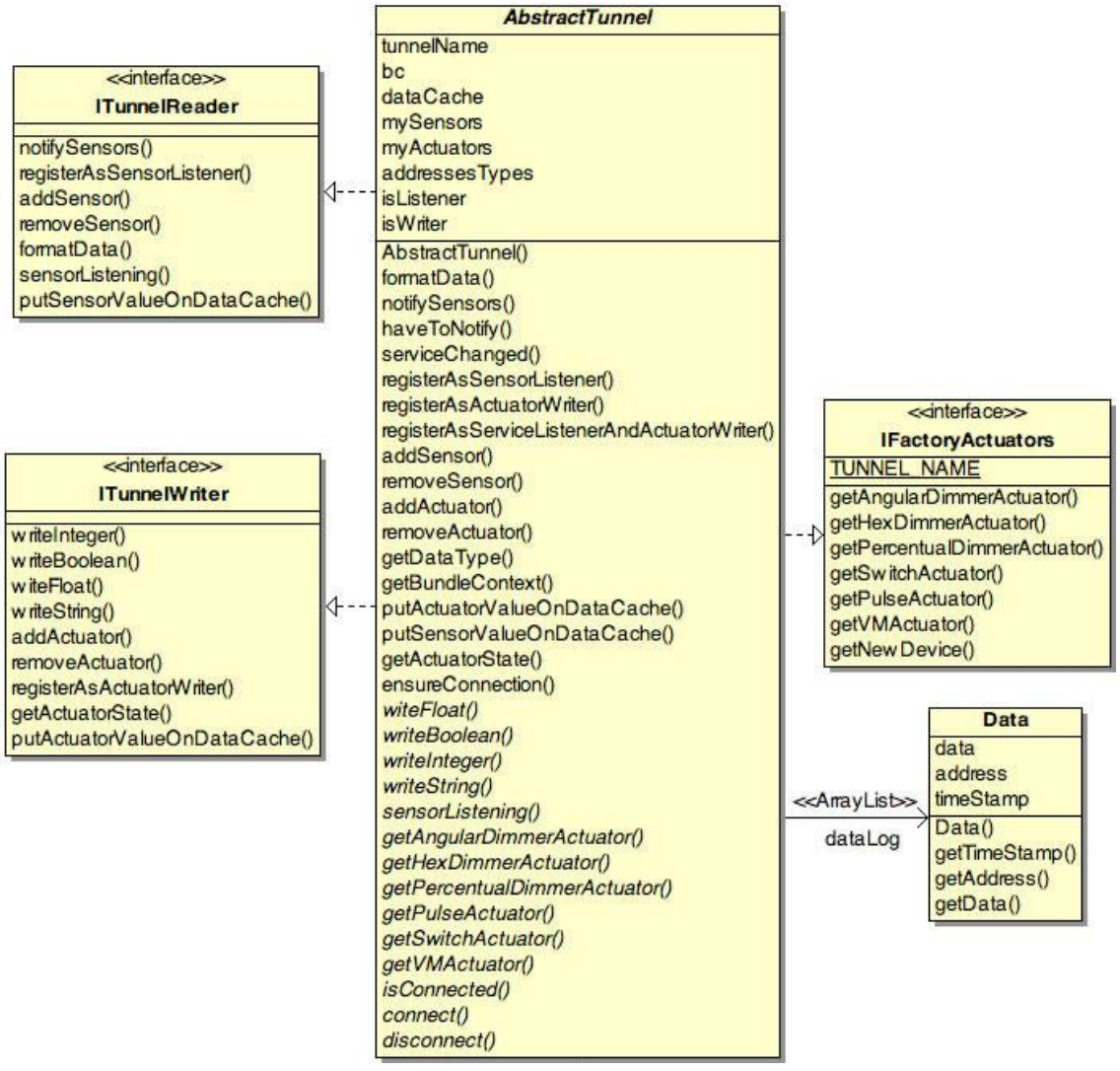

Ilustración 9 – Diagrama de clases del túnel

KNX/EIB

¿QUÉ ES UNA INSTALACIÓN KNX/EIB?

Una instalación KNX/EIB está formada por dispositivos domóticos (como bien pueden ser persianas, interruptores, iluminación, aire acondicionado…) que interactúan entre sí para dar un funcionamiento satisfactorio al usuario.

EIB es un sistema descentralizado, controlado por sensores, con transmisión de datos recibidos vía puerto serie (USB), con el objetivo de controlar y gestionar las funciones técnicas de una vivienda o edificio.

¿QUÉ ES EIBD?

EIBd (EIB *daemon*) es un servicio que, al ejecutarse sobre en un ordenador con sistema operativo Linux conectada por USB a una instalación física KNX/EIB, establece un enlace entre ésta y un puerto TCP que podrá ser utilizado por otras máquinas (o por ella misma) para interactuar con la instalación.

6. IMPLEMENTACIÓN Y SOLUCIÓN TECNOLÓGICA.

En este apartado se explicará con detalle la implementación del proyecto.

En este caso en concreto se trata de una implementación en lenguaje JAVA de los drivers de unos dispositivos domóticos (puestos como ejemplo) para que los Binding Providers puedan hacer uso de ellos.

Los dispositivos que se han montado físicamente para este proyecto se dividen en dos grupos, tales como Actuadores y Sensores. La diferencia entre ambos es que mientras que los actuadores no almacenan el estado anterior al actual, los sensores si, y en función de eso pueden actuar de forma distinta.

ACTUADORES

- Notificación visual
- Electroválvula
- Bombilla 1
- Bombilla 2
- Alarma
- Bombilla gradual
- Luz general

SENSORES

- Movimiento 1
- Movimiento 2
- Contacto 1
- Contacto 2
- Termostato
- Luminosidad
- Temperatura
- Estación meteorológica, formada por:

- Luminosidad
- Temperatura
- Lluvia
- Viento

Todos estos dispositivos están conectados a un PC por USB, y éste controla el ciclo de vida de cada uno de ellos, así como su interacción. Dentro del PC existe lo que se llama el Tunel (explicado en el Contexto Tecnológico) que soporta esta conexión y siempre interactúa con ella, de forma que los dispositivos físicos se comuniquen con el PC y viceversa. Esta comunicación se establece gracias a los drivers que soportan los componentes, y que permiten que los dispositivos sean reconocidos como tales. Estos drivers son el objetivo de este proyecto, mediante los cuales se podrá comunicar la arquitectura OSGi con los dispositivos físicos.

A modo de ejemplo, mostraremos el código del driver de la Alarma, el activador más simple que hay y explicaremos con más detalle qué se ha hecho. Numeraremos las líneas para facilitar su posterior referencia.

```
1. package alarma;

2. import java.util.Hashtable;
3. import java.util.Vector;

4. import org.osgi.framework.BundleActivator;
5. import org.osgi.framework.BundleContext;
6. import org.osgi.framework.ServiceReference;
7. import org.osgi.framework.ServiceRegistration;

8. import es.upv.pros.tunnel.interfaces.IDevice;
9. import es.upv.pros.tunnel.tools.Tools;

10.      public class Activator implements BundleActivator {

11.      Vector<ServiceRegistration> driversServiceRegistrations=new
   Vector<ServiceRegistration>();
12.      public void start(BundleContext context) throws Exception {
         Hashtable<String, Object> props=new Hashtable<String,
   Object>();
```

```
13.      props.put(IDevice.DEVICE_ADDRESS, "1/1/1");
14.      props.put(IDevice.DEVICE_DATA_TYPE,
   Boolean.class.getName());
15.      props.put(IDevice.DEVICE_DESCRIPTION, "Alarma");
16.      props.put(IDevice.DEVICE_ID, "AS");
17.      props.put(IDevice.DEVICE_LOCATION, "DSIC-LAB-103-MAQUETA");
18.      props.put(IDevice.DEVICE_MANUFACTURER, "Jesus");
19.      props.put(IDevice.DEVICE_MODEL, "R23");
20.      props.put(IDevice.DEVICE_NAME, "Alarma");
21.      props.put(IDevice.DEVICE_PROFILE,
   IDevice.DEVICE_PROFILE_EIB_ACTUATOR);
22.      props.put(IDevice.DEVICE_TUNNEL, "MAQUETA_LAB_103_DSIC");
23.      props.put(IDevice.DEVICE_TYPE,IDevice.DEVICE_TYPE_SWITCH);
24.      props.put(IDevice.DEVICE_SUBTYPE, "ON_OFF");
25.      props.put(IDevice.DEVICE_OTHER_PROPERTIES, new
   Hashtable<String, Object>());
26.      driversServiceRegistrations.add(Tools.createAndRegisterDriv
   er(context, props));

27.      }

28.      public void stop(BundleContext context) throws Exception {
29.      Tools.unregisterDrivers(driversServiceRegistrations);}

30.      }
```

Ilustración 10 – Código del driver de la alarma

Las primeras líneas (líneas 2-9) indican los paquetes que se utilizan para definir el paquete de alarma. Acto seguido, se define la clase Activator, como requisito de todo bundle (líneas 10-30), dentro de la cual están los métodos start() (líneas 12-27) y stop() (líneas 28-30) que definen su inicio y su parada. Esta clase hace uso de un vector de objetos de tipo ServiceRegistration (línea 11) que permite, cada vez que se inicie una alarma, registrarla dentro de ese vector y saber cuáles son las alarmas activas en cada momento (línea 26).

El método stop() en cambio lo que hace es quitar del registro la alarma activa, puesto que acaba de pararse.

En el caso de los sensores, la llamada de la línea 26 se substituye por otra que crea el sensor como tal:

```
new Sensor(context, props, new Hashtable<String, Object>(),
ISensor.NOTIFY_ALWAYS);
```

En esta llamada cabe destacar el cuarto parámetro, que puede ser de dos tipos y según se indique, el sensor funcionará de una forma u otra:

- `ISensor.NOTIFY_ALWAYS`: Indica que siempre que haya un cambio en el sensor, idependientemente de si es distinto al anterior o no, se notificará.

- `ISensor.NOTIFY_ON_CHANGE`: Indica que sólo cuando haya un cambio en el sensor distinto al anterior, se notificará.

PROPIEDADES

- ✓ `DEVICE_ADDRESS`: Dirección lógica del dispositivo.

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. Así, cada dirección se divide en área, línea dentro del área, y número de dispositivo, de la siguiente manera:

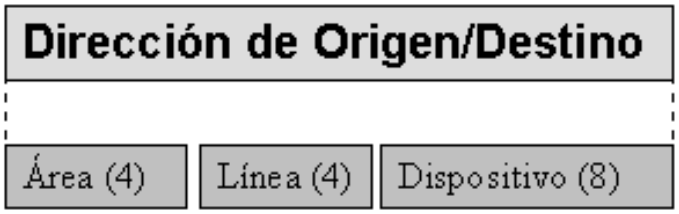

Ilustración 11 – Dirección física de un dispositivo

A pesar de esto, las direcciones que se utilizan aquí son las lógicas, denominadas también 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.

Esta dirección tendrá siempre la misma estructura, que en el caso del ejemplo de alarma será 1/1/1 (grupo principal / grupo intermedio / subgrupo). Para el resto de dispositivos, la relación sería la que sigue.

DISPOSITIVO	DEVICE_ADDRESS
Movimiento 1	1/4/7
Movimiento 2	1/4/8
Contacto 1	1/4/4
Contacto 2	1/4/9
Termostato	1/4/1
Luminosidad	1/4/5
Temperatura	1/4/6
Estación Meteorológica – Luminosidad	0/4/1
Estación Meteorológica – Temperatura	0/4/2
Estación Meteorológica – Lluvia	0/4/3
Estación Meteorológica - Viento	0/4/4
Notificación visual	1/0/0
Electroválvula	0/0/10
Bombilla 1	1/0/2
Bombilla 2	1/0/1
Bombilla Gradual	1/0/5
Luz general	0/0/1

Ilustración 12 – Relación Dispositivo– Dirección del dispositivo

- ✓ `DEVICE_DATA_TYPE`: Aquí se define el tipo de datos que enviará el dispositivo cada vez que se active / desactive. En el caso de todos los actuadores se tratará siempre de un Booleano (activado / desactivado), pero en el caso de los sensores dependerá de cada uno.

SENSOR	DEVICE_DATA_TYPE
Movimiento 1	Booleano
Movimiento 2	Booleano
Contacto 1	Booleano
Contacto 2	Booleano
Termostato	Flotante (Float)
Luminosidad	Entero
Temperatura	Flotante (Float)
Estación Meteorológica – Luminosidad	Entero
Estación Meteorológica – Temperatura	Flotante (Float)
Estación Meteorológica – Lluvia	Booleano
Estación Meteorológica - Viento	Entero (Velocidad)

Ilustración 13 – Relación Sensor – Tipo de dato del dispositivo

- ✓ `DEVICE_DESCRIPTION`: Aquí se indica una breve descripción del dispositivo en cuestión.

- ✓ `DEVICE_ID`: Aquí se indica el identificador de cada dispositivo. En el caso de la alarma es 'AS', pero para el resto de dispositivos es según se indica en la siguiente tabla.

DISPOSITIVO	DEVICE_ID
Movimiento 1	PD1
Movimiento 2	PD2
Contacto 1	WC1
Contacto 2	DC1
Termostato	CT1
Luminosidad	BR1
Temperatura	CT2
Estación Meteorológica – Luminosidad	MS_BRIGHT
Estación Meteorológica – Temperatura	MS_TEMP
Estación Meteorológica – Lluvia	MS_RAIN
Estación Meteorológica - Viento	MS_WIND
Notificación visual	AL
Electroválvula	EV
Bombilla 1	L1

Bombilla 2	L2
Bombilla Gradual	DIM
Luz general	GENERAL

Ilustración 14 – Relación Dispositivo – Identificador

- ✓ `DEVICE_LOCATION`: Aquí se indica la localización donde está ubicado el dispositivo. Por defecto, y para todos los dispositivos de este contexto, utilizaremos "DSIC-LAB-103-MAQUETA"

- ✓ `DEVICE_MANUFACTURER`: Aquí se indica el creador del driver, en este caso, soy yo.

- ✓ `DEVICE_MODEL`: Aquí se indica el modelo del dispositivo. No es significativo para el desarrollo de los drivers.

- ✓ `DEVICE_NAME`: Aquí se indica el nombre del dispositivo, en este caso, "Alarma"

- ✓ `DEVICE_PROFILE`: Aquí se indica el tipo de dispositivo que es: en el caso de los actuadores se indica mediante 'IDevice.DEVICE_PROFILE_EIB_ACTUATOR', mientras que si es sensor se indica mediante 'IDevice.DEVICE_PROFILE_EIB_SENSOR'

- ✓ `DEVICE_TUNNEL`: Aquí se indica el nombre del túnel que enlaza el PC con los dispositivos físicos, que para todos los dispositivos de este proyecto será "MAQUETA_LAB_103_DSIC"

- ✓ `DEVICE_TYPE`: Aquí se indica el tipo de dispositivo ante el cual nos encontramos. En el caso de los sensores siempre será 'IDevice.DEVICE_TYPE_SENSOR', pero en el caso de los actuadores dependerá de cada uno, según la tabla siguiente.

ACTUADOR	DEVICE_TYPE
Notificación visual	IDevice.DEVICE_TYPE_SWITCH
Electroválvula	IDevice.DEVICE_TYPE_SWITCH
Bombilla 1	IDevice.DEVICE_TYPE_SWITCH
Bombilla 2	IDevice.DEVICE_TYPE_SWITCH
Bombilla Gradual	IDevice.DEVICE_TYPE_SWITCH
Luz general	IDevice.DEVICE_TYPE_PERCENTUAL_DIMMER

Ilustración 15 – Relación Actuador – Tipo de dispositivo

- ✓ `DEVICE_SUBTYPE`: Aquí se puede indicar un subtipo de dispositivo. Por defecto los sensores no tendrán, y los actuadores tendrán el valor "ON_OFF"

- ✓ `DEVICE_OTHER_PROPERTIES`: Aquí se pueden indicar otras propiedades de interés según el dispositivo. En ninguno de estos dispositivos se gastarán estas propiedades, porque con las que tenemos nos bastan, así que se le asignará un valor vacío mediante 'new Hashtable<String, Object>()'

7. CONCLUSIONES

Las conclusiones que se obtienen de este proyecto son positivas, puesto que nunca había estado trabajando con este tipo de arquitecturas, y menos con aplicaciones orientadas a la domótica. A pesar de tratarse de una arquitectura pesada y algo complicada de entender, se ve que se pueden obtener muchos beneficios de esta tecnología.

8. REFERENCIAS

- **ARQUITECTURA: CALIMERO**

http://calimero.sourceforge.net/

http://eibcontrol.sourceforge.net/

- **TENOLOGIA: OSGI**

http://www.osgi.org

http://www.javahispano.org/contenidos/es/manual_de_osgi_por_roberto_montero/

- **DOCUMENTACIÓN TECNICA APORTADA POR EL DSIC**