UNIVERSIDAD POLITÉCNICA DE CARTAGENA

Departamento de Tecnología Electrónica

DESARROLLO DE SISTEMAS DOMÓTICOS UTILIZANDO UN ENFOQUE DIRIGIDO POR MODELOS

TESIS DOCTORAL

Manuel Jiménez Buendía

Ingeniero en Automática y Electrónica Industrial

Directores

Dr. D. Pedro Sánchez Palma

Dr. D. Andrés Iborra García

2009

Daría todo lo que sé, por la mitad de lo que ignoro. Descartes.

Los sabios buscan sabiduría, los necios creen haberla encontrado. Napoleón Bonaparte.

Agradecimientos

Es imposible recordar a todas las personas que han aportado voluntaria o involuntariamente su granito o montaña de arena a la realización de este trabajo, pero sería injusto no nombrar a los que han tenido una participación directa y muy importante.

En primer lugar, mi más sincero agradecimiento a mis directores Pedro Sánchez Palma y Andrés Iborra García (mucho más que mis directores de Tesis), sin su confianza, apoyo personal y profesional, empuje constante y dedicación no habría sido capaz de de llevar a cabo esta Tesis Doctoral. Gracias Pedro y Andrés. Asimismo quiero agradecer a Andrés la oportunidad de integrarme en el Grupo de Investigación División de Sistemas e Ingeniería Electrónica (DSIE). Gracias a la gestión y coordinación que realiza, el Grupo DSIE es un excelente entorno de trabajo.

A todos los miembros del Grupo de Investigación DSIE que han aportado su colaboración en este y otros proyectos relacionados. Quiero dar especialmente las gracias Paqui Rosique, con la que he colaborado estrechamente para desarrollar esta Tesis y que recoge el testigo para continuar con el trabajo realizado. Ánimo, que ya queda menos camino por recorrer. También quiero agradecer a Cristina Vicente Chicote y Diego Alonso Cáceres sus consejos y aportaciones desde su amplia experiencia en el "meta-mundo", y a José Alfonso Vera su colaboración en numerosos proyectos en el campo de la domótica.

A Natxo Matías Maestro, Carlos Fernández Valdivielso y Patxi Arregui de la Universidad Pública de Navarra, por sus aportaciones y colaboración en mis inicios con la domótica, allá por el año 2000.

Ya fuera del ámbito académico, quiero agradecer a mi familia y amigos su presencia y apoyo sin pedir nada a cambio durante estos años. Gracias Eva, por tu entrega desinteresada y por comprender la dedicación que este Trabajo de Tesis ha requerido.

Y finalmente, gracias a ti, lector, sin cuya presencia esto carecería de importancia.

Resumen

Esta Tesis Doctoral continúa la línea de investigación en el desarrollo de sistemas reactivos iniciada por el Grupo de Investigación DSIE de la Universidad Politécnica de Cartagena hace diez años, y parte de los resultados de otra Tesis anterior, en la que se desarrolló el metamodelo de componentes V^{3}Studio para recoger conceptos de sistemas reactivos. En trabajos previos se ha utilizado este metamodelo de componentes en los dominios de la robótica, la visión artificial y las redes inalámbricas de sensores y actuadores para obtener modelos independientes de la plataforma de ejecución y facilitar la confluencia e integración de diferentes dominios.

En este trabajo se completan las aportaciones anteriores proponiendo una metodología para el desarrollo de aplicaciones en el campo de la domótica, un dominio de los sistemas reactivos que integra numerosos servicios y sistemas relacionados con la gestión de viviendas y edificios. En el desarrollo de los sistemas domóticos se ven involucradas diversas materias como la automatización, tecnologías de la información, gestión de redes o la programación de microprocesadores. En la actualidad, el proceso de diseño en el campo de la domótica es similar al empleado en otros sistemas reactivos que interactúan con el entorno. En todos ellos es necesaria la intervención de un especialista del dominio que tiene una amplia experiencia en la plataforma sobre la que se realizará la implementación. Además, en la mayoría de los casos, el diseño se realiza prácticamente desde cero y se requiere un gran esfuerzo para la generación del código en el lenguaje de programación que se vaya a utilizar, conduciendo a soluciones a medida que rara vez son reutilizadas. Estos y otros muchos problemas plantean la necesidad de una mejora en el enfoque utilizado en el proceso de desarrollo.

Para solucionar los problemas asociados al proceso tradicional de desarrollo de aplicaciones domóticas, en esta Tesis se ha desarrollado (1) un lenguaje específico del dominio domótico para recoger los requisitos de las aplicaciones con un alto nivel de abstracción; (2) un metamodelo para darle soporte y las restricciones necesarias para garantizar la coherencia de los modelos y (3) una metodología que utiliza de forma conjunta el paradigma de *desarrollo dirigido por modelos* y el *lenguaje específico de dominio*. Para ello, se ha realizado una propuesta para la generación de código en la plataforma KNX/EIB y se han definido las reglas de transformación necesarias para hacer evolucionar los modelos entre los diferentes niveles de abstracción con el objetivo de obtener el código ejecutable. Como demostración de la viabilidad del enfoque se incluye un caso de estudio en el que se recogen los requisitos mediante el lenguaje específico propuesto.

Summary

This Thesis continues the research line in the development of reactive systems initiated by the DSIE research group of the Technical University of Cartagena ten years ago and starts from the results of a previous Thesis, in which a component metamodel, called V^{3}Studio, was designed to collect the requirements of reactive systems. This metamodel is intended (1) to be a point of confluence for reactive systems and (2) to obtain platform independent models. It has been previously used in the domains of robotics, computer vision and wireless sensor and actuator networks.

The present Thesis completes previous contributions proposing a new methodology and an associated environment to provide full support for the cycle of software development for home automation systems. The development of home automation systems involves areas such as information technology, automation, network management and microprocessors programming. At present, the design process in home automation is similar to the one employed in other reactive systems. In all of them a domain expert, with a wide background in the implementation platform, is required. Furthermore in most of cases the design is started from scratch and the effort to obtain the programming code is high, dealing to custom solutions which are rarely reused. These and many other problems raise the need for a new approach to improve the development process.

To solve the problems associated with the traditional development process in home automation this Thesis proposes (1) a *domain specific language* for the home automation domain to collect application requisites with a high abstraction level; (2) a metamodel to support the language and restrictions to guarantee well formed models and (3) a methodology which combines the newly *Model-Driven Engineering* paradigm together with the domain specific language. To achieve these goals a code generation approach for the KNX/EIB platform has been proposed and the required transformation rules have been defined. These rules are used to transform models from the highest abstraction level to the intermediate component model and, later, to platform models in order to produce executable code. Finally a case study example has been proposed to validate the language and the methodology.

Índice General

Índice de Figuras

Índice de Tablas

CAPITULO 1

Planteamiento y Objetivos

Este capítulo expone, en primer lugar, la motivación que ha llevado al desarrollo de este trabajo de Tesis junto con los objetivos que se persiguen. En segundo lugar se presenta la estructura de esta memoria, describiendo de manera concisa el contenido.

1.1 Motivación

Los sistemas reactivos han despertado un especial interés en la Ingeniería del *Software* por su alta complejidad e interacción con el entorno físico que les rodea. Es por ello que se hace especialmente difícil desarrollar software de *calidad* y de manera eficiente en estos ámbitos. Entre los sistemas reactivos se pueden enumerar sectores de la ingeniería tan variados como la robótica, la visión artificial, las redes de sensores inalámbricos (WSAN) y la domótica. Para el desarrollo de estos sistemas tradicionalmente se han utilizado criterios como la funcionalidad del sistema, la experiencia previa del diseñador y otros requisitos no funcionales como el coste máximo asumible. La mayor limitación de esta forma de proceder es la dificultad de conseguir artefactos software reutilizables, prefiriendo, por regla general, una solución eficiente y totalmente a medida, antes que diseñar soluciones más generales para facilitar su reutilización. Como consecuencia de esto, cada nuevo sistema debe construirse prácticamente desde cero, aunque su lógica y estructura sean casi idénticas a la de otros desarrollados previamente pero implementados sobre plataformas diferentes. Por ello, la utilización de nuevos métodos, técnicas y herramientas de la Ingeniería del *Software*[1] se plantea como una necesidad en este campo.

Por otra parte, la aparición de *MDE (Model Driven Engineering*) [Selic 03] y en especial de la propuesta *MDA (Model Driven Architecture)* [MDA 03] del *OMG (Object Management Group)* ha propiciado un nuevo impulso en la investigación del uso de lenguajes específicos de dominio *(DSLs*) y la generación automática de código.

En MDE el objetivo principal es construir software a partir de **modelos**, desplazando así el uso tradicional del código fuente como protagonista principal de los procesos de desarrollo. En este contexto, los lenguajes específicos de dominio toman un papel importante como lenguajes de modelado que permiten describir el sistema de una forma fácil e intuitiva.

Los DSLs aportan conceptos de un dominio de aplicación y permiten identificar primitivas de alto nivel al recoger conocimiento de expertos en el dominio [Czarnecki 05]. Aunque se han utilizado ampliamente durante años, ha sido recientemente cuando se ha comenzado a realizar un estudio sistemático de ellos, impulsado por el planteamiento de la metodología MDE. En este nuevo contexto los DSLs facilitan el trabajo en las primeras etapas de diseño y, además, el enfoque MDE ayuda a reducir el coste de desarrollo de los DSLs. Se trata, por tanto, de una unión sinérgica que permite mejorar significativamente el proceso de desarrollo del software.

[1] A lo largo de esta memoria se hace uso de términos y vocabulario de origen anglosajón (algunos castellanizados) teniendo su justificación en la amplia difusión de la lengua inglesa en las distintas áreas de la ingeniería. En algunos casos, existe el equivalente en castellano (por ejemplo, "implantar" en vez de "implementar"). En otros, se adopta una palabra del castellano con un significado distinto al original (como "instancia" aceptado por la comunidad informática como la traducción de "*instance*").

Gracias a este nuevo enfoque, se puede abordar la creación de herramientas para el control y gestión de sistemas **reactivos** desde una perspectiva mucho más eficaz, obteniendo herramientas más interoperables y fáciles de mantener mediante técnicas que incrementen el nivel de abstracción. Un ejemplo de ello es el de los sistemas **domóticos**, un caso particular de sistema reactivo capaz de interactuar con el entorno que le rodea aportando servicios de gestión energética, seguridad, comunicaciones y confort, quedando todos estos automatismos '*integrados*' completamente en la vivienda, dotándola de una cierta '*inteligencia*' con el fin de mejorar la vida cotidiana.

Estos sistemas se desarrollan en la actualidad con técnicas de bajo nivel de abstracción y sin ninguna metodología que permita recoger los requisitos del sistema de forma independiente de la plataforma, por lo que se necesita un grado de especialización muy elevado y la reutilización de los artefactos *software* empleados es mínima.

La espectacular penetración y previsible crecimiento tanto en número de dispositivos como en complejidad de estos sistemas ha agudizado la necesidad de disponer de herramientas y metodologías adecuadas para abordar el desarrollo de los sistemas domóticos de una forma más eficiente.

En la actualidad prácticamente no existen lenguajes específicos en el campo de la domótica que permitan una captura de requisitos con cierto nivel de abstracción e independencia de la plataforma. Se han encontrado algunas propuestas basadas en MDA que utilizan lenguajes de modelado (UML) [Voelter 07][Muñoz 07] poco intuitivos y alejados de los conceptos manejados por los expertos del dominio domótico. Las otras propuestas encontradas se corresponden con herramientas comerciales dependientes de la plataforma. Las más conocidas son *ETS* (*Engineering Tool Software*) y *LonMaker*, que son específicas de las plataformas *KNX/EIB* y *Lonworks, respectivamente*.

Por ello, se establece como punto de partida de este trabajo la aplicación de métodos de Ingeniería del *Software* (MDE y DSLs) para solucionar los problemas asociados al proceso de diseño actual en el ámbito de la domótica.

1.2 Entorno de Desarrollo de la Tesis

La Tesis que aquí se expone se enmarca dentro de los trabajos realizados por el Grupo de Investigación *División de Sistemas e Ingeniería Electrónica*[2] (DSIE) de la Universidad Politécnica de Cartagena (UPCT). El DSIE nació en 1999 como Grupo de Investigación multidisciplinar e integra profesores e investigadores de los departamentos de Tecnología Electrónica (DTE), Ingeniería de Sistemas y Automática y Tecnologías de la Información y las Comunicaciones de la

[2] http://www.dsie.upct.es

UPCT. El DSIE desarrolla su labor de investigación, entre otras, en las siguientes áreas de tecnológicas dentro del ámbito de los sistemas reactivos:

- Sistemas de control y robótica para aplicaciones industriales.
- Robots de servicio.
- Sistemas de inspección visual automatizados.
- Tecnología electrónica para robótica y visión artificial.
- Redes de sensores (*Wireless Sensor Networks*).
- Automatización en viviendas y edificios (Domótica).

Esta Tesis Doctoral profundiza en la aplicación de nuevas técnicas de la Ingeniería del *Software* para la mejora en el proceso de desarrollo de sistemas reactivos de automatización en viviendas y edificios (Domótica). Esta línea fue abierta por el Dr. D. Diego Alonso Cáceres, que en su Tesis Doctoral [Alonso 08a] propuso la utilización de MDE para el desarrollo de sistemas reactivos, aunque se centrado en el dominio de los robots de servicio. No obstante, la arquitectura de componentes diseñada por el Dr. Alonso se ha empleado en el nivel independiente de plataforma dentro de la propuesta metodológica realizada en este trabajo de Tesis como se detallará más adelante.

Los trabajos que han posibilitado la consecución de esta Tesis se enmarcan en el proyecto MEDWSA (*Marco conceptual y tecnológico para el desarrollo de software de sistemas reactivos*) de la Comisión Interministerial de Ciencia y Tecnología (CICYT TIN2006-15175-C05-02). MEDWSA es uno de los subproyectos del proyecto coordinado META (*Models, Environments, Transformations and Applications*), que se ha venido desarrollando entre los años 2007-2009. El objetivo del proyecto MEDWSA es la definición de un marco conceptual y tecnológico para el desarrollo de sistemas reactivos que aproveche las ventajas de las tendencias actuales del desarrollo dirigido por modelos. A la vista del objeto de la presente Tesis Doctoral, MEDWSA supone un marco de trabajo ideal para finalizar su desarrollo y validarla.

1.3 Objetivos

El objetivo de esta Tesis es definir las bases teóricas y prácticas necesarias para establecer una nueva metodología para abordar el ciclo de vida completo para el diseño de sistemas domóticos siguiendo un enfoque dirigido por modelos (MDE) junto con la utilización de lenguajes específicos de dominio (DSLs) como soporte a la definición de los requisitos de las aplicaciones. De forma más específica se pueden concretar los objetivos siguientes:

- Estudiar las características y particularidades de los sistemas domóticos como dominio singular dentro del ámbito de los sistemas reactivos. Este estudio permitirá seleccionar de manera adecuada las plataformas tecnológicas de destino dentro del proceso de desarrollo.

- Revisar el enfoque metodológico basado en el desarrollo dirigido por modelos (MDE) y, más en concreto, la propuesta MDA del OMG. Utilizando este enfoque se seleccionarán las herramientas de desarrollo de entre las disponibles en la actualidad para soportar su utilización en el ámbito de los sistemas reactivos.
- Proponer un lenguaje específico de dominio (DSL) que permita recoger los requisitos de una aplicación domótica de forma gráfica e intuitiva para el desarrollador a la vez que agilice el proceso de desarrollo de aplicaciones, facilite la verificación y aumente el nivel de abstracción en la definición de los requisitos del sistema. Para ello se definirá, por un lado, la sintaxis abstracta del lenguaje mediante un metamodelo y las restricciones necesarias y, por otro, la sintaxis concreta (metáfora gráfica) mediante elementos apropiados para este dominio.
- Establecer una metodología que conjugue de manera sinérgica el lenguaje específico de dominio con la propuesta MDA del OMG para obtener código ejecutable de manera automática o semiautomática para diferentes plataformas a partir de los requisitos definidos en el DSL. La creación de esta metodología, fundamental en el desarrollo del este trabajo de Tesis, implica la consecución de los siguientes objetivos:
  - Adoptar y adaptar un modelo arquitectónico de componentes como nivel independiente de la plataforma que permita la integración con otros sistemas reactivos en el ámbito de trabajo del Grupo de Investigación DSIE.
  - Adoptar un marco formal para la representación de las transformaciones entre los distintos niveles conceptuales.
  - Establecer correspondencias entre conceptos del DSL y el modelo de componentes independiente de plataforma, que se traducirán en la definición de transformaciones automatizadas.
  - Seleccionar conjunto de herramientas y estrategias que permitan la generación de código específico de plataforma. Para ello será necesario seleccionar plataformas objetivo para validar la viabilidad de la generación de código para posteriormente extender estas técnicas a otras tecnologías domóticas.
  - Establecer correspondencias entre el modelo intermedio de componentes y las soluciones específicas de fabricante de cara a la generación de código ejecutable.
  - Integrar todo el conocimiento anterior en un conjunto de herramientas unificadas que se desarrollarán en esta Tesis para dar soporte automático al desarrollo de sistemas domóticos en el marco MDE.
- Por último, y para validar la aplicación del enfoque propuesto en este trabajo de Tesis, se desarrollará un caso de estudio utilizando el DSL creado para aplicaciones domóticas.

1.4 Estructura de la Tesis

Esta memoria se ha dividido en un total de seis capítulos. Además, el documento contiene dos anexos con información adicional y una última sección en la que se recogen las citas bibliográficas. La estructura detallada de la Tesis Doctoral es la siguiente:

Capítulo 1: Introducción

En este capítulo se realiza una breve introducción en la que presenta la motivación y objetivos perseguidos en el desarrollo esta Tesis. Asimismo, se ha presentado el marco de trabajo en el que se encuadra esta investigación.

Capítulo 2: Sistemas Domóticos

Este capítulo describe el dominio de trabajo de esta Tesis mediante la descripción de los aspectos más importantes de los sistemas domóticos. La domótica es un campo de la automatización y las comunicaciones que está tomando un protagonismo creciente en los últimos años, y resulta imposible su comprensión sin un estudio detallado de sus características

Capítulo 3: Desarrollo Dirigido por Modelos (MDE)

Este capítulo, fundamental para la comprensión de la metodología subyacente al desarrollo de esta Tesis, describe el enfoque MDE (Model Driven Engineering) de desarrollo de software dirigido por modelos, que utiliza los modelos como artefacto principal en todo el proceso de diseño software. MDE proporciona una nueva teoría de desarrollo y una serie de herramientas que soportan su aplicación con el fin de aumentar el nivel de abstracción para la realización del software y obtener de forma automática o semiautomática las diferentes representaciones del mismo y el código final ejecutable. Además, se revisa el estado del arte de la utilización del enfoque MDE en el desarrollo de sistemas reactivos, prestando especial atención a su aplicación a los sistemas domóticos.

Capítulo 4: Lenguaje Específico de Dominio

El objetivo de este capítulo es presentar un lenguaje específico para el dominio de la domótica. Para ello se ha realizado un estudio detallado de las ventajas e inconvenientes del uso de DSLs, así como de los enfoques que se pueden utilizar a la hora de abordar su diseño para un campo de aplicación concreto. Además, se estudian los escasos DSL existentes para el dominio de la domótica indicando sus cualidades y deficiencias. También se analizan y organizan aquellos aspectos y conceptos más relevantes del dominio domótico de cara a la definición del DSL. Siguiendo la filosofía MDE, el DSL se ha definido conforme a un metamodelo que especifica las relaciones entre los conceptos del domino. Finalmente se revisan las herramientas utilizadas para

crear el DSL en un marco de gestión de modelos basado en la propuesta MDA, y el enfoque utilizado para abordar la definición de la semántica.

Capítulo 5: Desarrollo de Sistemas Domóticos Dirigido por Modelos

En este capítulo se presenta ***HAbitATION*** (*development of Home Automation ApplicaTIons using a mOdel driveN approach*), una metodología para abordar el ciclo de vida completo de desarrollo de sistemas domóticos siguiendo un enfoque dirigido por modelos (MDE), basándose en la propuesta MDA del OMG y en la utilización de lenguajes específicos de dominio (DSLs) como soporte para la definición de los requisitos de las aplicaciones. Para ello se recoge la problemática asociada al proceso tradicional de desarrollo para sistemas domóticos, se propone la metodología basada en MDA que soluciona las deficiencias detectadas y se profundiza en las características propias de una plataforma específica como tecnología para obtener una implementación del sistema. Asimismo se describen las transformaciones necesarias para, a partir de los modelos creados con el DSL, obtener los modelos intermedios y se proponen las herramientas y enfoque para la obtención del código final específico de plataforma.

Capítulo 6: Conclusiones y Trabajos Futuros

En este último capítulo se resumen las aportaciones realizadas en esta Tesis Doctoral y los resultados obtenidos. Además se proponen las líneas de investigación más interesantes de cara a consolidar y ampliar el trabajo realizado.

Anexo 1: Caso de Estudio

En este anexo se expone un caso de estudio en el ámbito de la domótica consistente en la automatización de una sala de juntas o reuniones utilizando el DSL. Este caso de estudio permite demostrar y validar el funcionamiento del DSL en una aplicación real.

Anexo 2: Ejemplo de Transformaciones entre Niveles CIM - PIM

Este anexo incluye un ejemplo completo donde se presentan de forma detallada las transformaciones aplicadas para obtener el modelo de componentes intermedio (nivel PIM) a partir de un modelo sencillo creado con el DSL. De esta manera se puede verificar el correcto funcionamiento de las transformaciones, así como su viabilidad.

2

Sistemas Domóticos

En el presente capítulo se hace una revisión de los aspectos más importantes de los sistemas domóticos, cuyo desarrollo se ha definido como objetivo en el planteamiento de esta Tesis Doctoral.

La domótica es un campo de la automatización y las comunicaciones que está tomando un protagonismo creciente en los últimos años. Sólo conociendo en detalle la tecnología y estándares existentes se puede tener una visión global de la complejidad de estos sistemas y, lo que es más importante, de la necesidad de un lenguaje específico del dominio domótico.

En el primer apartado se define con mayor precisión el significado del término domótica y otros similares. A continuación se describen sus ámbitos de aplicación y la domótica como disciplina integradora. En el tercer apartado se exponen los componentes y arquitecturas que caracterizan a las tecnologías utilizadas. El apartado cuatro revisa la normalización existente en la actualidad, y en el siguiente se estudian las tecnologías comerciales más extendidas, mostrando las características más relevantes para poder comprender el proceso de desarrollo con las mismas. Finalmente se da una perspectiva de la rápida evolución del mercado de la domótica hasta la fecha y las previsiones de expansión en el futuro.

2.1 Introducción

En los últimos años, las tecnologías de la información y las comunicaciones se están integrando en el hogar y la vida cotidiana a gran velocidad [Ryan 89][Hernández 97]. Este proceso ha dado lugar a un nuevo tipo de sistemas reactivos: los sistemas domóticos.

Para la aparición de esta nueva tecnología han sido fundamentales varios factores: por un lado la disponibilidad del elemento base para el desarrollo de la informática en los últimos tiempos (el microprocesador) y por otro, la convergencia entre la informática y las telecomunicaciones, junto con la necesidad cada vez mayor de información a todos los niveles [Tidd 94].

Asimismo, en su evolución ha tenido una gran repercusión la definición paralela de arquitecturas de comunicación de datos en el ámbito de la automatización industrial: los conocidos buses de campo, con los que los sistemas domóticos presentan grandes similitudes. De hecho, es muy difícil establecer una separación clara entre ambos campos, ya que la literatura existente incluye a muchos de los protocolos para redes de control domótico dentro de las redes de automatización industriales.

Desde el punto de vista etimológico, los orígenes del término nos llevan a Francia (uno de los países pioneros en Europa en este campo), donde se acuñó el término *"Domotique"* como contracción de *"domus"* (vivienda) y automática. En nuestro país, el término **domótica** se definía en 1988 como *"el concepto de vivienda que integra todos los automatismos en materia de seguridad, gestión de la energía, comunicaciones, etc."* [Larousse 08]. La definición de Vivienda Domótica o Inteligente presenta múltiples versiones y matices, y son diversos los términos utilizados en distintos idiomas: ***casa inteligente*** *(smart home),* ***automatización de viviendas*** *(home automation),* ***domótica*** *(domotique),* ***sistemas domóticos*** *(home systems),* etc. Hasta hoy se conocen múltiples definiciones de domótica, de las que cabe destacar las siguientes:

- La nueva tecnología de los automatismos de maniobra, gestión y control de los diversos aparatos de una vivienda, que permiten aumentar el confort del usuario, su seguridad y el ahorro del consumo energético.
- La informática aplicada a la vivienda. Agrupa el conjunto de sistemas de seguridad y de la regulación de las tareas domésticas destinadas a facilitar la vida cotidiana automatizando sus operaciones y funciones.
- Conjunto de servicios de la vivienda garantizado por sistemas que realizan varias funciones, los cuales pueden estar conectados entre sí y a redes interiores y exteriores de comunicación. Gracias a ello se obtiene un notable ahorro de energía, una gestión eficaz técnica de la vivienda, una buena comunicación con el exterior y un alto nivel de seguridad.

Pero quizás una de las más completas es la que se recoge en [ITC-BT-51], que dice: *"Sistemas de Automatización, Gestión de la Energía y Seguridad para Viviendas y Edificios:*

Son aquellos sistemas centralizados o descentralizados, capaces de recoger información proveniente de unas entradas (sensores o mandos), procesarlas y emitir órdenes a unos actuadores o salidas, con el objeto de conseguir confort, gestión de la energía o la protección de personas, animales y bienes. Estos sistemas pueden tener la posibilidad de acceso a redes exteriores de comunicación, información o servicios, como por ejemplo, red telefónica conmutada, servicios INTERNET, etc.".

Existe aún hoy cierta polémica en cuanto a la idoneidad del término domótica ya que el objeto de esta disciplina no es únicamente la vivienda sino cualquier tipo de edificación. Por ello, se han creado diversos términos para distinguir el alcance de las domótica según el sector de aplicación [Nozick 88]:

- Domótica, para el sector doméstico (aunque hoy día se ha generalizado además para el sector edificios).
- Inmótica, para el sector terciario (automatización de edificios como hoteles, hospitales, oficinas, etc.).
- Urbótica, para las ciudades. Control de la iluminación pública, gestión de semáforos, telecomunicaciones, medios de pago, etc.

En la actualidad la arquitectura de un sistema domótico, al estar prácticamente basada en una red más o menos compleja de comunicaciones, nos lleva a tratarlo como una **Red Domótica**, a la cual conectamos dispositivos de lo más variado y a los que podemos acceder desde cualquier punto de la red. Con esta idea, se puede definir una Red Domótica como una **instalación inteligente capaz de interactuar con el medio que le rodea**.

2.2 Tipos de Aplicaciones de la Domótica

Las aplicaciones desarrolladas en domótica ofrecen la posibilidad de gestionar un sistema inteligente mediante la modificación local o remota de los parámetros de la instalación. Para ello ofrecen una serie de servicios realizados por un conjunto de automatismos o dispositivos con cierto grado de inteligencia (basados en microcontroladores) dirigidos a la consecución de cuatro objetivos básicos (véase la Figura 2-1):

- **Gestión Energética y Recursos:** regulación de la climatización, gestión de los consumos de cada electrodoméstico y de la potencia contratada, control del suministro de recursos como electricidad, gas y agua, etc.
- **Seguridad:** custodia y vigilancia frente a la intrusión, la inundación, el fuego, los escapes de gas, etc.
- **Comunicaciones:** comunicación interna del sistema, telecontrol y telemetría, SMS, señales acústicas, etc.
- **Confort:** automatización de tareas repetitivas, programaciones horarias, escenarios luminosos, riego automático, etc.

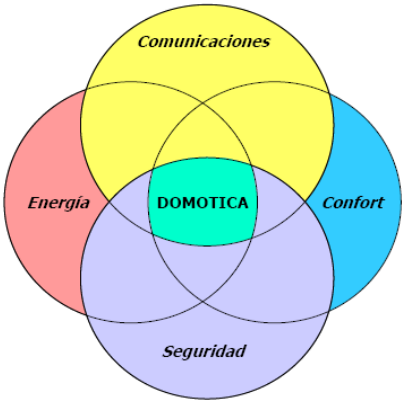


Figura 2-1. Aplicaciones de la domótica.

Las fronteras entre estos cuatro objetivos son difusas y en muchos casos un mismo dispositivo favorece el logro de varios objetivos a la vez, lo cual, por otra parte, economiza la instalación. Es precisamente esta filosofía de **integración** la que da realmente significado a la domótica, ya que de otro modo estaríamos hablando de automatizaciones independientes. Es decir, la instalación domótica va más allá de la mera automatización de una vivienda o edificio, ya que integra el control de una serie de sistemas y el uso que se hace de ellos.

A continuación se tratarán en detalle los aspectos más importantes de estas cuatro áreas de aplicación básicas. En la Figura 2-2 se muestran algunos ejemplos de elementos para cada una de las áreas.

2.2.1 Gestión Energética y Recursos

La finalidad es satisfacer las necesidades del hogar minimizando el consumo energético. En este control se pueden distinguir tres aspectos diferenciados:

- Regulación con la que se pueda obtener la evolución del consumo energético de la vivienda o edificio.
- Programación para establecer distintos parámetros, como temperatura según horarios, días de la semana, mes, etc.
- Optimización para minimizar el consumo. El aprovechamiento de la energía y reducción de su consumo es uno de los apartados más importantes en la instalación de un sistema domótico, puesto que revierte a medio y largo plazo en su amortización, además de estar muy ligados al concepto de confort. Las acciones destinadas a reducir el consumo están íntimamente relacionadas con la integración de todos los dispositivos de la vivienda en el sistema. Estas acciones son del tipo [Waks 91]:
  - Aprovechamiento de las franjas de tarificación de valle para hacer trabajar aquellos equipos que lo permitan (por ejemplo, aprovechamiento de tarifas nocturnas en función de las necesidades programadas).

- Reducción del consumo para climatización fuera de las horas de trabajo normales.
- Detección de fuentes de pérdidas en sistemas de climatización (por ejemplo, suspensión del funcionamiento en estancias donde se detecten ventanas abiertas).
- Reducción del consumo para climatización o iluminación en ausencia de individuos en las estancias mediante la detección automática de presencia.
- Actuación sobre automatismos de persianas para el aprovechamiento de la luz solar.

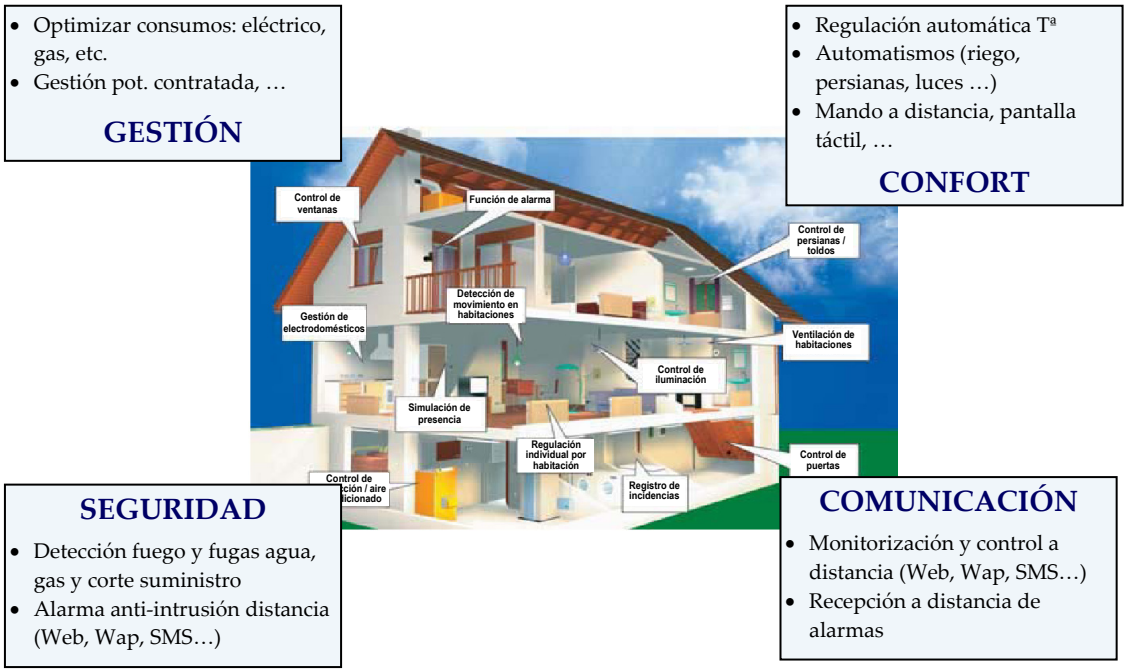


Figura 2-2. Ejemplos de automatización en una vivienda [Konnex 08].

2.2.2 Confort

El concepto de confort va dirigido principalmente a las instalaciones de climatización, ventilación y calefacción (HVAC, *Heating, Ventilating, and Air Conditioning*), aunque también se incluyen en este campo los sistemas de audio y vídeo, control de iluminación, riego y jardines, mando a distancia y todo aquello que contribuya al bienestar y la comodidad de las personas que utilicen las instalaciones. En los sistemas de HVAC es donde mayores inversiones se están realizando, pues además de abarcar una gran parte del consumo energético, están presentes en casi todas las instalaciones y son la primera contribución. Se hace necesario que el control de estos sistemas esté lo más distribuido posible, esto es, que cada habitación, local o recinto, disponga de control individual.

Entre los sistemas destinados al confort cabe destacar, además de los mencionados:

- Control por infrarrojos de los distintos automatismos.

- Automatización de riego de jardines.
- Apertura automática de puertas.
- Centralización y supervisión de todos los sistemas de la vivienda en dispositivos como pantallas táctiles o centros multimedia.
- Accionamiento automático de distintos sistemas en función de datos del entorno, como la recogida automática de toldos, bajada de persianas en caso de tormenta o viento excesivo, encendido automático de luces en zonas de paso (pasillos, escaleras), etc.
- Información de presencia de correo en el buzón.

2.2.3 Seguridad

La seguridad es la función más solicitada e instalada, aunque de manera individualizada (no integrada) y puede integrar múltiples aplicaciones (véase la Figura 2-3), sobre todo si se encuentra integrada dentro de un sistema domótico. Se distinguen dos áreas básicas: seguridad de personas y seguridad de bienes.

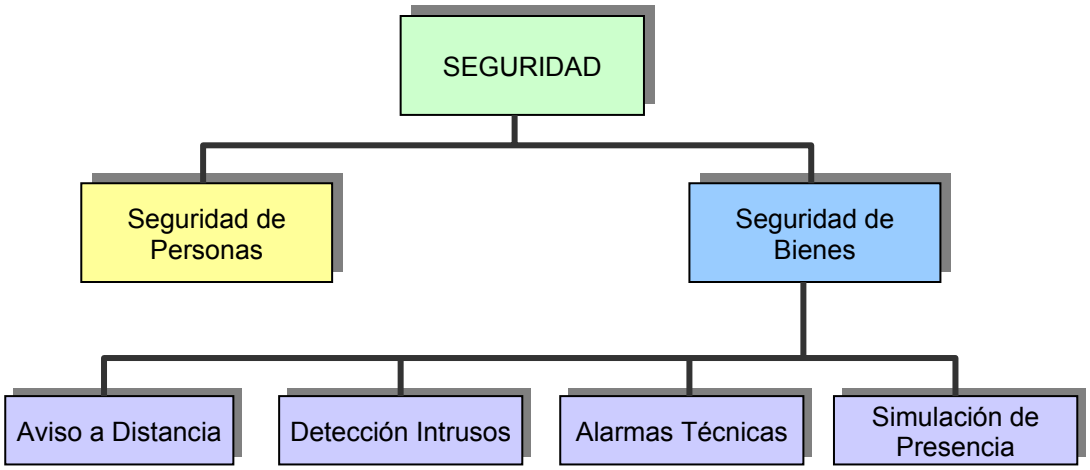


Figura 2-3. Funciones de seguridad.

En la seguridad de personas se incluyen tareas como:

- Alumbrado automático en zonas de riesgo por detección de presencia (escaleras, etc.) para evitar accidentes domésticos.
- Desactivación de enchufes de corriente para evitar contactos.
- Manipulación a distancia de interruptores en zonas húmedas.
- Emisión de avisos telefónicos a números prefijados en caso de necesidad de ayuda urgente.
- Detectores de fugas de gas o de agua que cierren las válvulas de paso a la vivienda en el caso de producirse escapes.

- Alarmas de salud. En el caso de personas con necesidades especiales (ancianos, personas discapacitadas) se dispone de pulsadores cuya activación genera un aviso a una central receptora, un familiar o un hospital para solicitar ayuda sanitaria urgente.

En lo referente a la seguridad de bienes, las funciones principales son:

- Avisos a distancia. En ausencia del usuario se emiten avisos en caso de alarma (bien acústicos o telefónicos).
- Detección de intrusos. Incluye la instalación de diversos sensores como sensores volumétricos para detección de presencia, sensores de hiperfrecuencia para cristales rotos o sensores magnéticos para apertura de puertas y ventanas.
- Alarmas técnicas. En este apartado son típicas la detección de incendios, detección de fugas de agua y gas y detección de cortes de suministro eléctrico. También se pueden realizar acciones correctivas (por ejemplo, si se detecta escape de gas entonces se debe cortar el suministro).

2.2.4 Comunicaciones

En el área de comunicaciones existen numerosas posibilidades en función del tipo de instalación. La aparición de nuevas tecnologías en el campo de las comunicaciones y redes de transmisión de datos, y el hecho de que los sistemas domóticos avanzados se basan en el empleo de estos tipos de redes, hacen de éste un campo fértil para la investigación y el desarrollo de nuevas arquitecturas y sistemas de integración [Matías 99].

En la actualidad no existen soluciones integrales que abarquen todas las redes y protocolos que podemos encontrar en una vivienda o edificio, sino que encontramos diferentes tecnologías que han de integrarse para proporcionar los servicios deseados (véase la Figura 2-4). Las redes que son internas a la instalación se denominan HAN (*Home Area Network*) y son de tres tipos:

- Red domótica o de control de dispositivos de la vivienda, como electrodomésticos, puntos de luz, persianas, pulsadores, etc. Suelen utilizar protocolos de redes de control domótico como KNX/EIB [KNX 04], Lonworks [Byoug 00], CEBus [EIA 92], X10 [X10 08], etc.
- Red de Datos. Típicamente redes de tipo Ethernet cableadas o inalámbricas.
- Red multimedia, como la televisión por cable o telefonía.

Debido a la ausencia de soluciones integradas se hace necesario el uso de **pasarelas residenciales**, dispositivos capaces de interconectar las diferentes redes internas y externas de la vivienda (véase la Figura 2-4).

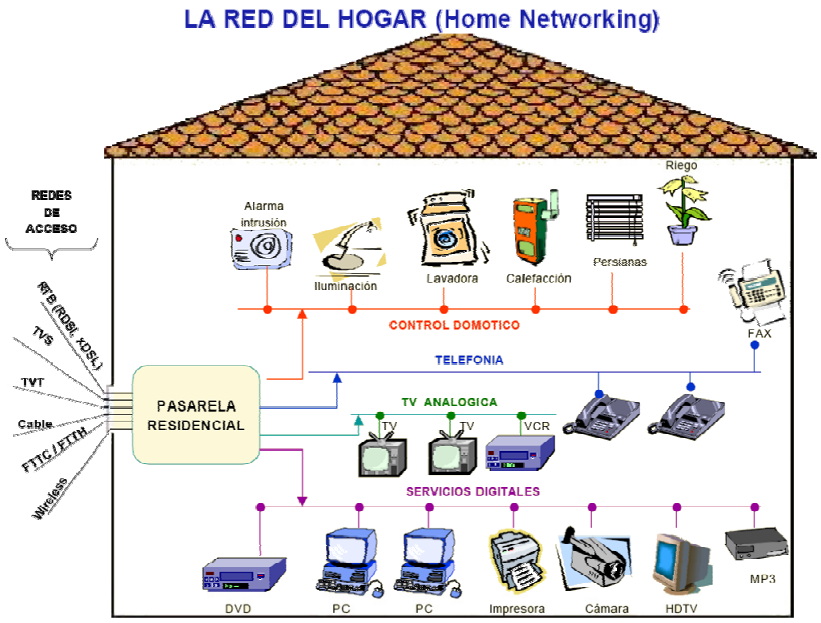


Figura 2-4. Redes del hogar [Libro Blanco 03].

Entre las posibilidades de telecomunicación según el tipo de edificio, destacan:

- Sistemas de comunicación en el interior: megafonía, difusión de audio/video, intercomunicadores, etc.
- Sistemas de comunicación con el exterior: telefonía básica, video-conferencia, e-mail, Internet, TV digital, TV por cable, fax, radio, transferencia de datos (X25, ATM), etc.
- Comunicaciones externas propias de la vivienda. Mensajes de alarma como fugas de gas, agua, etc., y telecontrol del sistema domótico a través de la línea telefónica o redes de área extensa (Internet).

De entre todas ellas, las que mayor auge están teniendo en los últimos años, desde el punto de vista de aportaciones de investigación e implantación de nuevas tecnologías, son las iniciativas de telecontrol del sistema domótico desde el exterior. En este sentido se pueden destacar trabajos como:

- Desarrollo de iniciativas abiertas para la implementación de servicios domóticos en pasarelas residenciales [OSGi 08].
- Control de instalaciones domóticas mediante protocolo TCP/IP utilizando el lenguaje *html* o *applets* del lenguaje Java, para la teleoperación y monitorización de sistemas domóticos en edificios [Ott 99][Nunes 00][Werthschult 01] [Wernetges 07][Acuña 08].
- Control de instalaciones domóticas mediante dispositivos móviles (teléfonos móviles, PDA, etc.), mediante servicio de mensajes cortos (SMS, *Short Message Service*), protocolo de aplicaciones inalámbricas (WAP, *Wireless Application Protocol*) y otras tecnologías.

- Aplicación de sistemas de encriptación y autentificación en el acceso remoto a instalaciones domóticas a través de Internet [Fernández 01], para asegurar la privacidad y seguridad de los datos en el acceso a través de redes públicas.
- Aplicación de técnicas de 'Diseño Para Todos' *(Design for all*) para facilitar la interacción con el entorno doméstico de personas mayores o con algún tipo de discapacidad [Vera 00][Vera 01].

2.3 Tecnología en Sistemas Domóticos

Se ha definido la Red Domótica como una instalación inteligente capaz de interactuar con el medio que le rodea. Esta red se compone de una serie de dispositivos que **detectan cambios de estado en las variables del entorno y los transmiten a otros elementos para que puedan actuar en consecuencia, en función de unas reglas establecidas por el usuario del sistema**. Resulta por tanto evidente, que para ello serán necesarios una serie de dispositivos, interconectados por algún tipo de medio de transmisión, y organizados según una arquitectura para la comunicación entre ellos.

2.3.1 Componentes

Los elementos que componen un sistema domótico se clasifican en los siguientes tipos:

- **Sensores** o dispositivos de entrada: captan cambios en determinados parámetros físicos del entorno y los convierten en señales eléctricas que son enviadas a los elementos del control para que tomen las decisiones.
- **Controladores** (o nodos): dispositivos capaces de recibir y procesar información, y comunicarse con otros controladores o dispositivos.
- **Actuadores**: son dispositivos de salida capaces de recibir órdenes de un controlador y realizar una acción (encendido/apagado, subida/bajada de una persiana, apertura/cierre de electroválvulas, etc.).

Dependiendo de la tecnología o solución utilizada hay equipos que son a la vez controladores, sensores y actuadores, o combinaciones de ellos. Por ejemplo, un termostato-controlador de estancias (KNX o Lonworks), incluye sensores para medir temperatura, teclas y la inteligencia necesaria para generar órdenes de control sobre los aparatos de climatización. Este hecho abre infinitas posibilidades a la hora de combinar funcionalidad y, en consecuencia, repercutirá en las decisiones que se adopten a la hora de definir un lenguaje específico de dominio (véase el capítulo 4).

2.3.2 Soportes de Transmisión

El soporte de transmisión es el empleado por los diferentes elementos de control para intercambiar información. A continuación se presentan los medios físicos utilizados en domótica, que no difieren de los utilizados en redes de transmisión de datos.

2.3.2.1 Líneas de Distribución de Energía Eléctrica (Corrientes Portadoras)

Es una de las alternativas más utilizadas cuando se trata de instalaciones en viviendas ya construidas ya que al aprovechar la propia red eléctrica de baja tensión se evita el problema que supone la instalación de un cableado dedicado, facilitando así enormemente el conexionado de los dispositivos.

No obstante, sus desventajas son numerosas. Hay que utilizar una interfaz electrónica para superponer la señal de datos de alta frecuencia a la de la red de baja tensión (230V/50Hz en España), lo que eleva considerablemente el coste de los equipos. Existen numerosas fuentes de interferencias debido a la presencia de un elevado número de armónicos, y además hay que filtrar las líneas de la instalación eléctrica para evitar que la red de alta tensión afecte a la de baja, lo que también implica un coste adicional.

Teóricamente, la impedancia de la red de baja tensión es del orden de los 50Ω a 100KHz entre fase y neutro, pero esta impedancia disminuye cuando aumenta el número de cargas conectadas, lo que provoca una degradación importante de las señales y limita la velocidad de transmisión.

Por lo tanto, el sistema de corrientes portadoras no es el más aconsejable en instalaciones domóticas, aunque puede ser adecuado cuando las necesidades del sistema no impongan requerimientos muy exigentes en cuanto a la velocidad y fiabilidad de transmisión.

El espectro de la red de baja tensión está dividido en varias zonas con usos reservados (véanse Figura 2-5 y Tabla 2-1), lo que supone una limitación más en su uso.

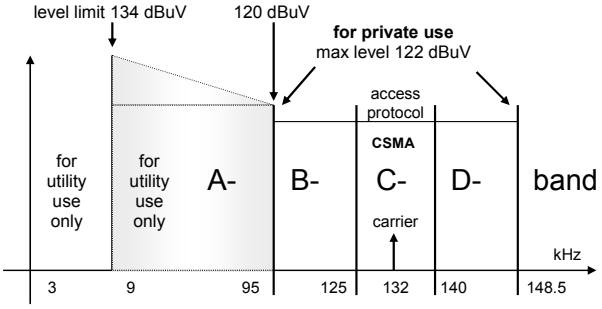


Figura 2-5. Bandas de frecuencia PLC [EN 50065BCD]. Adaptado de [Klaus 01]

Banda	Rango frecuencia	Uso
A	9-95KHz	Asignado a empresas de suministro eléctrico. En esta banda no es necesario ningún método de acceso al medio (MAC).
B	95-125KHz	Libre. Requiere MAC. Diseñada para usarse en aplicaciones como intercomunicadores.
C	125-140KHz	Libre. Requiere MAC.
D	140-148.5KHz	No requiere MAC.

Tabla 2-1. Uso de las bandas de frecuencia PLC [EN 50065BCD][Klaus 01].

Sistemas tan populares como X10 [X10 08], basado en corrientes portadoras, tienen una velocidad de transmisión de 50 ó 60 bits por segundo (dependiendo de la frecuencia de la red eléctrica). Otros protocolos más fiables que utilizan corrientes portadoras, como la versión PL (*Power Line*) de KNX/EIB, alcanzan velocidades de 1200 bits por segundo.

2.3.2.2 Soportes Metálicos

Se pueden distinguir tres tipos:

- Cables paralelos, como los utilizados tradicionalmente en telefonía. En aplicaciones domóticas su uso se limita a la conexión de sensores a controladores o conexiones eléctricas auxiliares.
- Par trenzado. Tal como ocurre con las redes de comunicación de datos (como las de tipo Ethernet), es el medio físico más habitual en sistemas de cableado dedicado por su bajo coste.
- Cable coaxial. En viviendas se utiliza fundamentalmente para la distribución de señales de televisión y radio.

En sistemas domóticos, el soporte más empleado es el cable de par trenzado [Mercahome 04], motivado fundamentalmente por la amplia oferta de productos en las tecnologías más importantes (Konnex y Lonworks), que basan su oferta de productos en sus versiones con este tipo de medio físico.

2.3.2.3 Fibra Óptica

En los últimos años, el empleo de la fibra óptica ha sido creciente en las redes de comunicación de datos, motivado por sus excepcionales características: gran inmunidad al ruido, escasa atenuación que permite transmisiones a grandes distancias y un ancho de banda muy elevado.

En el ámbito de los sistemas domóticas su uso es muy escaso, motivado por el elevado coste del cableado la instalación de la fibra. Su uso se restringe a la interconexión de grandes sistemas con cableado sobre par trenzado, en los niveles más altos de la topología. No obstante, existen algunos diseños de redes domóticas empleando fibra, como los recogidos en los trabajos de investigación [Arregui 97][Kojima 93][Muñiz 96].

2.3.2.4 Transmisión sin Hilos

Para la transmisión sin hilos existen dos alternativas: infrarrojos y radiofrecuencia.

Infrarrojos

Están muy extendidos para la transmisión de información en el interior de estancias desde dispositivos móviles, fundamentalmente para el control de equipos de audio y vídeo.

Al tratarse de un medio de transmisión óptico, es inmune a radiaciones electromagnéticas, pero es necesaria visibilidad entre el emisor y el receptor. Existe

además un problema de normalización, ya que no existe compatibilidad entre los distintos emisores y receptores de diferentes fabricantes.

Este medio se utiliza con mucha frecuencia en aplicaciones domóticas como soporte de apoyo a determinadas aplicaciones, como la integración de equipos de audio/vídeo o empleo de mandos a distancia infrarrojos para el control de determinadas funciones (encendido de luces, control de persianas, llamada de escenas, etc.). Algunos de los estándares existentes definen el empleo de este medio.

Radiofrecuencia

El empleo del medio inalámbrico para la comunicación entre los sensores y actuadores que constituyen una red domótica ha sido estudiado en diversos trabajos, bien mediante la adaptación de un estándar existente como el francés Télédomotis [Kauffman 00], o bien a través de nuevas propuestas de protocolos y sistemas de transmisión [Fujieda 00][Hakem 02][Tsang 03]. Asimismo, existen soluciones comerciales propietarias como Enocean [Enocean 08] y Hometronic de Honeywell [Honeywell 02], pero su coste es elevado y la implantación en el mercado muy escasa.

El empleo de este medio de transmisión, que en principio puede parecer idóneo para la implantación de sistemas domóticos en viviendas construidas, se ha visto frenado por sus inconvenientes: elevada sensibilidad a perturbaciones electromagnéticas producidas por los equipos domésticos, necesidad de sistemas de seguridad en la transmisión y escaso alcance en las transmisiones.

Se espera que la verdadera revolución en el sector de la domótica inalámbrica se produzca con la introducción de las redes de sensores y actuadores inalámbricas (WSAN). Iniciativas como la especificación del estándar IEEE 802.15.4 [Callaway 02] para WSAN definen la automatización de viviendas como uno de sus principales ámbitos de aplicación.

2.3.3 Arquitecturas

La arquitectura de un sistema domótico especifica el modo en que se van a conectar los distintos componentes de la instalación: sensores, actuadores y controladores.

El empleo de diferentes filosofías de cableado e incluso distintos tipos de red, hace que existan diferencias notables en parámetros como la complejidad del cableado, velocidad de transmisión, vulnerabilidad, gestión de la red, tasa de fallos, etc.

Desde el punto de vista de dónde reside la inteligencia del sistema domótico, existen dos tipos distintos de sistemas domóticos:

- **Sistemas Centralizados**: En este tipo de sistemas toda la información relativa a la detección y actuación se tratan en un punto único que es la unidad central. El controlador centralizado recibe información de múltiples sensores y, una vez

procesada, genera las órdenes oportunas para los actuadores. Toda la información de detección y actuación se tratan en este único punto (véase la Figura 2-6).

Figura 2-6. Ejemplo de arquitectura centralizada en sistema domótico.

El cableado que se suele utilizar en estos casos sigue una estructura en estrella cuyo centro es la unidad central de control y no existe comunicación entre sensores y actuadores. Entre sus ventajas destacan:

- Bajo coste ya que ningún elemento necesita módulos especiales de direccionamiento ni interfaces para distintos buses.
- Instalación sencilla y posibilidad de utilizar una gran variedad de elementos comerciales.
- Requerimientos mínimos.

Y entre sus inconvenientes:

- Flexibilidad limitada ya que las reconfiguraciones son costosas.
- Poca robustez puesto que si cae el módulo central cae todo el sistema.
- Mayor longitud de cableado dada la topología, lo que incrementa el coste de la instalación y limita su uso en grandes instalaciones.

- **Sistemas Distribuidos**: En este caso no existe la figura del controlador centralizado sino que toda la inteligencia del sistema está distribuida por todos los módulos, sean sensores o actuadores (véase la Figura 2-7). Cada elemento dispone de capacidad para tratar la información que recibe y actuar en consecuencia de forma autónoma.

  La arquitectura distribuída es típica de los sistemas con topología en bus y se requiere un protocolo de comunicaciones. Todos los elementos disponen de un acoplador al bus con una interfaz de acceso compartido y técnicas de direccionamiento para que la recepción y el envío de información quede definida y el diálogo entre elementos asegurado. Es habitual, además, que se permitan cableados de topología libre, de manera que se facilita su instalación en la vivienda o edificio.

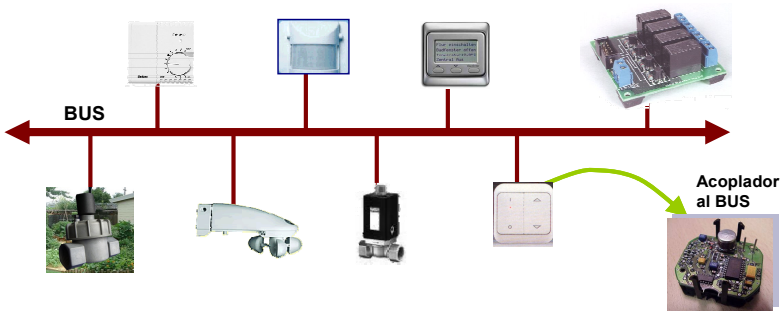


Figura 2-7. Ejemplo de arquitectura distribuida en sistema domótico.

Las principales ventajas de los sistemas distribuidos son:

- Alta flexibilidad y una gran facilidad para reconfiguraciones.
- Escalabilidad. Suelen ser adaptables a cualquier tamaño de instalación y las ampliaciones resultan sencillas.
- Posibilidad de tecnologías *plug & play* que simplifican mucho las instalaciones.
- Ahorro de cableado en la instalación, lo que reduce los costes, sobre todo en instalaciones y proyectos a gran escala.

Sus inconvenientes:

- Mayor precio de los componentes, dado el incremento de complejidad que conllevan por la necesidad de incluir los protocolos y técnicas de direccionamiento utilizados.
- Necesidad de compatibilidad entre los equipos y componentes.
- Oferta de productos restringida al protocolo que emplean para garantizar la compatibilidad entre ellos.

Hay sistemas que son de arquitectura distribuida en cuanto a la capacidad de proceso, pero no lo son en cuanto a la ubicación de los diferentes elementos de control, que se encuentran físicamente centralizados. También ocurre al contrario, hay sistemas que son de arquitectura distribuida en cuanto a que ubican los dispositivos de forma distribuida (por ejemplo disponen de módulos de entrada salida remota), pero ejecutan los procesos de control en uno o varios procesadores físicamente centralizados (los denominados sistemas **descentralizados**).

Hoy en día hay buenos sistemas centralizados y distribuidos, todos ellos con elevadas prestaciones. Ambas arquitecturas tienen sus ventajas y sus inconvenientes, lo cual, a priori, no ayuda a decidir cuál es la mejor solución para una vivienda.

2.4 Normalización

Una norma es un documento de **aplicación voluntaria** que contiene especificaciones técnicas basadas en los resultados de la experiencia y el desarrollo tecnológico [AENOR 08]. A la hora de elaborar una norma, debe existir un consenso entre todas las partes interesadas en la actividad objeto de la misma. Además, debe aprobarse por un Organismo de Normalización reconocido. A pesar de esta no obligatoriedad, determinadas disposiciones legales, que sí son de obligado cumplimiento, hacen referencia a normas, de modo que aplicando estas normas se estará de acuerdo con lo indicado por dichas disposiciones.

2.4.1 Organismos de Normalización

En la Tabla 2-2 se representan los distintos organismos de normalización, distinguiéndose por su ámbito de aplicación y el sector en el que trabajan. Cada uno de estos organismos está formado por distintos grupos de trabajo que agrupan temas en los que se elaboran las normas relacionadas:

ISO: *International Organization for Standardization* [ISO 08] / **IEC:** *International Electrotechnical Commission* [IEC 08].

La Comisión Electrotécnica Internacional está compuesta por 43 países a nivel mundial. Se organiza en comités técnicos (TC o JTC), subcomités (SC) y grupos de trabajo (WG). El del JTC 1 engloba a las "Tecnologías de la Información" (*Joint ISO/IEC Technical Committee* establecido en 1987). El Subcomité 25 (SC25) "Interconexión en la Tecnología de la Información" es el responsable de la interconexión en la tecnología de la información. Dentro de su campo de aplicación está la normalización de sistemas microprocesadores, así como de interfaces, protocolos y medios de interconexión asociados para equipos de tecnología de la información, generalmente para entornos comerciales y residenciales. Se excluye el desarrollo de normas para redes de telecomunicaciones e interfaces a redes de comunicación. Dentro de este subcomité, el grupo de trabajo 1 "Sistemas Electrónicos del Hogar" (WG1: *Home Electronic Systems*) se encarga de los sistemas electrónicos en viviendas. Por lo tanto, la normalización relativa a la domótica se encuentra recogida en el comité 1, subcomité 25, grupo de trabajo 1 (JTC1/SC25/WG1).

	General	Eléctrico	Telecomunicac.
Internacional	ISO	IEC	ITU
Europeo	CEN	CENELEC	ETSI
Nacional	AENOR		

Tabla 2-2. Organismos de normalización de interés en España por sector y ámbito de aplicación.

CEN: *European Committee for Standardization* [CEN 08].

El Comité Técnico 247 "Automatización de Edificios, Controles y Gestión de Edificios", se encarga de la normalización de automatización de edificios, controles y gestión de edificios y servicios para edificios residenciales y no residenciales.

Estas normas incluyen definiciones, requisitos, funciones y métodos de ensayo de los productos de automatización de edificios y sistemas para control automático de instalaciones de servicios en edificios.

Las medidas de integración primarias incluyen interfaces de aplicación, sistemas y servicios para asegurar una gestión técnica de edificios eficiente en cooperación con la gestión comercial y de infraestructuras del edificio.

Se excluyen de su campo de aplicación las áreas de automatización de edificios bajo la responsabilidad de otros comités de CEN/CENELEC.

CENELEC: *European Committee for Electrotechnical Standardization* [CENELEC 08].

El CENELEC está formado por 18 países europeos, 16 de los cuales están dentro del IEC. El Comité Técnico 205 (TC205) "Sistemas electrónicos para viviendas y edificios" se encarga de preparar normas para todos los aspectos de sistemas electrónicos domésticos y en edificios en relación a la sociedad de la información.

Más en detalle, prepara normas para asegurar la integración de un espectro amplio de aplicaciones y aspectos de control y gestión de otras aplicaciones en y entorno a viviendas y edificios, incluyendo las pasarelas residenciales a diferentes medios de transmisión y redes públicas, teniendo en cuenta todo lo relativo a compatibilidad electromagnética (EMC, *Electromagnetic Compatibility*) y seguridad eléctrica y funcional.

El TC205 no prepara normas de producto sino los requisitos de actuación necesarios y los interfaces de hardware y software necesarios. Las normas deberán especificar ensayos de conformidad. El TC205 realiza el trabajo en estrecha cooperación con los comités técnicos relevantes de CENELEC, CEN y ETSI.

ITU: *International Telecommunication Union* [ITU 08].

ETSI: *European Telecommunications Standards Institute* [ETSI 08].

AENOR: Asociación Española de Normalización y Certificación [AENOR 08].

En España, AENOR es el organismo de normalización y está formado por Comités Técnicos de Normalización y Subcomités. Estos grupos están formados por los entes interesados (Administración, empresas, universidades, etc.) y en ellos se elaboran y discuten los contenidos de las normas. La aportación de España hacia y desde los organismos europeos e internacionales se realiza siempre a través de AENOR como proposición consensuada del país.

Existen otros organismos de normalización nacionales fuera del ámbito europeo, tales como la **EIA** (*Electronic Industries Association*) en Estados Unidos, o la **EIAJ** (*Electronic Industries Association of Japan*) en Japón, que son las encargadas de la normalización en los citados países

2.4.2 Normativa y Disposiciones Legales

Como se ha dicho en la definición, una norma es de aplicación voluntaria y por tanto su cumplimiento no es obligatorio, son las **disposiciones legales** las que sí definen una obligación expresa de cumplimiento, que en algunas ocasiones se dirige a una o varias normas.

Podemos hacer una distinción entre las disposiciones legales europeas y las nacionales. Las primeras de ellas son elaboradas por la Comisión europea que las publica en el Diario Oficial de la Unión Europea (DOCE) y su fin último es armonizar las diferentes reglamentaciones nacionales. Cada uno de los estados miembros de la UE adapta estas disposiciones a su legislación, y en el caso concreto de España, las directivas nacionales se publican en el BOE en forma de Real Decreto.

El marco normativo actual no dispone de directivas específicas para el sector de la domótica que deban aplicarse en cualquier instalación. No obstante, las disposiciones legales, y por tanto de obligado cumplimiento, que tienen relación más o menos directa con el sector y que deben considerarse a la hora de hablar de productos y sistemas domóticos, se detallan en la Tabla 2-3.

<table>
<tr><th>Normas Técnicas</th><th>Disposiciones Legales</th></tr>
<tr><td>• Serie de Normas EN 50090 "Home and Building Electronic Systems (HBES)" e ISO/IEC 14543 "Home Electronic Systems (HES) Architecture"
• Serie de Normas EN/ISO 16484 "Building Automation and Control Systems (BACS)"
• Serie de Normas EN 14908 "Open Data Communication in Building Automation"</td><td rowspan="2">• Directivas europeas
▪ BT 2006/95/CE
▪ CEM 2004/108/CE

• Reglamentación Nacional
▪ CTE
▪ Reglamento de ICT
▪ REBT
▪ ITC-BT 51</td></tr>
<tr><td>• CWA 50487 "SmartHouse Code of Practice"
• EA 0026 "Instalaciones de Sistemas Domóticos en Viviendas"</td></tr>
</table>

Tabla 2-3. Normas y disposiciones legales relacionadas con la domótica.

Entre las normas técnicas destacan las series de normas de los tres protocolos de comunicación que están normalizados por Organismos de Normalización, y cuyo código es accesible y de libre utilización:

- Serie de Normas EN 50090 "*Home and Building Electronic Systems (HBES)*". Normas del protocolo Konnex, que engloba los tres protocolos europeos previos Batibus, EHS y EIB, y basa su funcionamiento en este último. El protocolo Konnex se encuentra recogido desde el año 2007 en las normas ISO/IEC 14543-3-X.
- Serie de Normas EN/ISO 16484 "*Building Automation and Control Systems (BACS)*", con las normas del protocolo BACNET.

- Serie de Normas EN 14908 "*Open Data Communication in Building Automation*", con las normas del protocolo LON (Lonworks).

Los dos documentos restantes que se han incluido en este apartado, no son estrictamente normas técnicas sino documentos de referencia, si bien el proceso seguido para su elaboración es muy similar al de una norma:

- CWA 50487 "*SmartHouse Code of Practice*". Documento elaborado a través de una reunión internacional del CENELEC con el objetivo de proporcionar una guía práctica para el diseño, instalación y mantenimiento de sistemas domóticos (*smart house*).
- EA 0026 "Instalaciones de Sistemas Domóticos en Viviendas. Prescripciones generales de instalación y evaluación". Documento elaborado por el Subcomité de normalización SC205 "Sistemas Electrónicos en Viviendas y Edificios" en estrecha colaboración con el CEDOM (Asociación Española de Domótica), que establece los requisitos mínimos que deben cumplir las instalaciones domóticas de Clase I para su correcto funcionamiento, así como las prescripciones generales para la evaluación de la aptitud en viviendas.

Los aspectos que sí son de obligado cumplimiento son los mencionados en las diferentes directivas y reglamentaciones que les afectan:

- Directivas europeas:
  - **Directiva 2006/95/CE de Baja Tensión**. Su finalidad es la de garantizar la seguridad en el empleo de cualquier material eléctrico.
  - **Directiva 89/336/CEE de Compatibilidad Electromagnética**. Cuyo objetivo es garantizar la protección de los equipos y las personas contra los problemas que puedan causar las perturbaciones electromagnéticas que provocan los dispositivos eléctricos y electrónicos. Esta disposición quedará derogada por la nueva directiva que entrará en vigor el 20 de julio de 2009 **2004/108/CE**.
- Reglamentación nacional:
  - **Código Técnico de la Edificación (CTE, RD 314/2006).** Tras entrar en vigor el 29 de marzo de 2007, sus principales objetivos son asegurar la calidad en la edificación y promover la sostenibilidad e innovación. Entre otros requisitos, la nueva normativa obliga a que los edificios construidos bajo su aplicación, cuenten con fuentes de energía renovables para la obtención de electricidad y agua caliente. Aunque la domótica no es obligatoria en las construcciones, colabora con el fin del CTE de conseguir edificios más eficientes desde el punto de vista energético, disminuyendo el consumo de energía.
  - **Reglamento de Infraestructuras Comunes de Telecomunicaciones (ICT, RD 401/2003).** Este reglamento deben cumplirlo todas las edificaciones sujetas a la ley de la propiedad horizontal y establece las especificaciones técnicas en

materia de comunicaciones para el interior de los edificios con la finalidad de garantizar a los ciudadanos el acceso a las telecomunicaciones (Radiodifusión sonora y Televisión terrestres y vía satélite, redes telefónicas RTC y RDSI, y redes de banda ancha por cable y radio). Del mismo modo que en el CTE, no se hace referencia expresa a la domótica, si bien es cierto que podría ser uno de los documentos más fácilmente ampliables para recoger la legislación en lo referente a servicios domóticos.

- **Reglamento Electrotécnico de Baja Tensión (REBT, RD 842/2002).** Actualmente, este reglamento es el que se considera como documento por excelencia para guiar el diseño de una instalación domótica, contemplando ésta como un caso particular de instalación eléctrica. De entre las 51 instrucciones que componen el REBT, cabe hacer especial mención de la instrucción **ITC-BT 51 "Instalaciones de sistemas de Automatización, gestión técnica de la energía y seguridad para viviendas y edificios"**, en la que se intentan establecer los requisitos específicos de una instalación domótica o inmótica. Con objetivo de facilitar el seguimiento de las instrucciones del REBT y completarlas al mismo tiempo, se han publicado unas guías de las instrucciones técnicas, que no son de obligado cumplimiento pero están en consonancia con las instrucciones a las que se refieren. En concreto la Guía de la ITC-BT 51 especifica, entre otros, los tipos de redes que pueden existir en una vivienda, los instaladores autorizados o la documentación que debiera proporcionarse con la instalación.

2.5 Tecnologías Existentes

En la actualidad existen numerosos sistemas domóticos comerciales, orientados a distintos segmentos del mercado. Desde el punto de vista de los sectores a los que van destinados, se pueden distinguir tres: viviendas ya construidas, casas de nueva construcción y grandes edificios (véase la Figura 2-8).

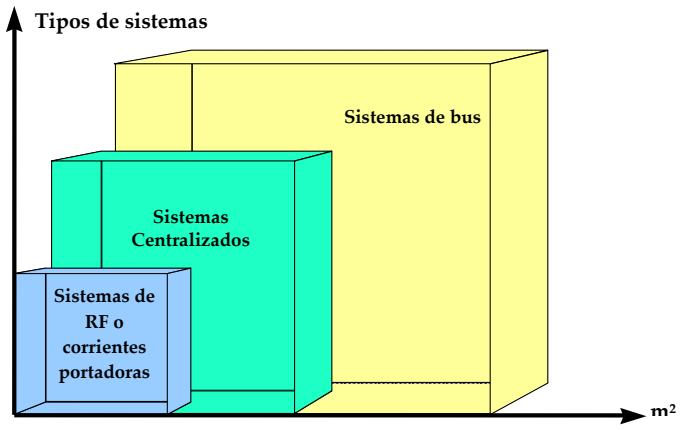


Figura 2-8. Sistemas domóticos en función del tamaño de la instalación.

En **viviendas construidas** existen tan sólo dos alternativas, el empleo de sistemas con transmisión por corrientes portadoras o bien por radiofrecuencia. La implantación de los sistemas por radiofrecuencia es muy escasa, debida, por una parte a la escasa oferta comercial, y por otra a los problemas que se derivan de su uso (poca fiabilidad, alcance limitado, etc.). La implantación de los sistemas por corrientes portadoras es mucho mayor, siendo el más instalado el americano X10 de *Home Systems*, debido a su antigüedad en el mercado y bajo coste, y el sistema CAD de Legrand en Europa. Sin embargo, existen versiones PL (*Power Line* o Corrientes Portadoras) de los sistemas Konnex y Lonworks (este último uno de los más fiables y avanzados del mercado) que están tomando cada vez más fuerza.

En **viviendas nuevas** la oferta es muy extensa, con sistemas que permiten cubrir todas las necesidades con topologías centralizadas o distribuidas y protocolos abiertos o propietarios. En viviendas pequeñas-medianas con requerimientos básicos son muy frecuentes los sistemas centralizados propietarios, como IHS (*Innovation House Control*), que en España comercializa Simón bajo la denominación de Simon VIS. Otros sistemas propietarios son Simon VOX (Simon), Sicov (ISDE ingeniería), Amigo (Merlin Gerin), Microdelta (Delta Dore), Vivimat (Dinitel), Cardio (Secant) y un largo etcétera.

Cuando los requerimientos son más exigentes y se trata de inmuebles de mayor tamaño, es cada vez más frecuente el uso de sistemas de bus distribuidos, especialmente los normalizados **Konnex** y **Lonworks**, cuyo abaratamiento y gran oferta de productos en los últimos años los está haciendo muy populares (véase el apartado "2.6 Evolución de la Domótica en el Mercado Español").

En el caso de los **edificios** las necesidades suelen ser mucho más complejas que en una vivienda. En este ámbito los sistemas de bus aventajan a los demás en cuanto a prestaciones, aunque es aún frecuente encontrar sistemas de control centralizado basados en autómatas de gama alta cuando la relación cableado/componentes lo permite. Hay que tener en cuenta que éstos han sido los sistemas tradicionalmente instalados para la gestión técnica de edificios, existentes antes de la aparición de tecnologías específicas para la domótica.

Los sistemas de tipo bus más instalados en Europa son **Batibus** de Merlin Gerin, **EHS**, y el sistema Konnex/EIB (KNX/EIB). En Estados Unidos, el sistema de bus más popular es **Lonworks** de Echelon, que en Europa está poco implantado, aunque recientemente se ha recogido en normas de la CENELEC. Otros sistemas aplicables en este tipo de instalaciones son CEBus de la EIA, EHS de EHSA, *Smart House* de la NAHB, y en el caso de los sistemas de control centralizado de gama alta autómatas de los principales fabricantes como Siemens, Omron, Schneider Electric, etc.

Por lo tanto, se puede concluir que las tecnologías domóticas más relevantes en la actualidad son, en el mercado americano, **CEBus, X-10 y Lonworks**, y en el europeo **KNX/EIB, Batibus y EHS**.

Los sistemas Europeos más importantes (Batibus, EIB y EHS) se han unido formando un consorcio para conseguir la compatibilidad entre ellos. En este proceso, denominado convergencia, se está impulsando el uso de EIB como tecnología base, por lo que es con esta tecnología (KNX/EIB) con la que se están realizando mayor número de instalaciones.

A continuación se detallan las características más importantes de estos sistemas.

2.5.1 CEBus

En Estados Unidos, la EIA (*Electronic Industries Association*) reconoció la necesidad de desarrollar un estándar acerca de los sistemas de comunicación de los hogares automatizados. En 1983 se organizó un comité que tuvo como fruto en 1988 un estándar (el *Home Automation Standard* IS-60) conocido como *Consumer Electronic Bus* (CEBus)[Douligeris 93]. El documento final, después de varias revisiones, estuvo disponible en 1992 [EIA 92] y cubre tanto las características eléctricas como los procedimientos de los módulos del sistema de comunicación.

La arquitectura del estándar CEBus sigue el modelo de referencia OSI (*Open Systems Interconnection*), ocupándose cada uno de los niveles de determinadas funciones de la red de comunicación. CEBus sólo utiliza cuatro de los siete niveles OSI: Físico, Enlace, Red y Aplicación. La interfaz entre los diferentes niveles del nodo CEBus está definida como un conjunto de primitivas de servicio, proporcionando cada nivel servicio al inmediatamente superior. En CEBus se diferencian tres áreas:

- El medio físico y la topología.
- El protocolo de comunicaciones (cómo acceder al medio y construir los mensajes).
- El lenguaje de programación (conjunto de acciones que se pueden efectuar en el sistema).

El protocolo y el lenguaje son comunes a todos los elementos CEBus, pero existen 6 medios físicos distintos:

1. Red eléctrica (PL).
2. Par trenzado (TP).
3. Infrarrojo (IR).
4. Radio frecuencia (RF).
5. Coaxial (CX).
6. Fibra óptica (FO).

La elección del medio se realiza en función de parámetros como el ahorro energético, comodidad, facilidad de instalación de los productos CEBus, seguridad, coste y sencillez del sistema.

En una instalación pueden coexistir diversos medios. Cada uno de ellos constituiría una subred local (*Local Medium Network*). Las subredes locales se conectan mediante encaminadores (*routers*).

CEBus engloba varios canales de comunicación: uno de control y varios de datos. En el canal de control se intercambian mensajes y órdenes para el control de los dispositivos de la instalación domótica. Los canales de datos se emplean para la transmisión de voz, música, TV, vídeo etc., y se asignan por solicitud mediante el canal de control. Por lo general, la distribución de las distintas señales se realiza de la siguiente manera:

- Señales de vídeo: mediante dos cables coaxiales, uno para las señales internas y otro para las externas.
- Señales de voz/datos: cuatro pares trenzados denominados TP0 a TP3 (TP0 se reserva para la alimentación de $18V_{dc}$).
- Resto de señales: a través de la red de BT, conectando equipos a enchufes estándar. Se utiliza una técnica de modulación con espectro ampliado de Intellon Corp.

La velocidad de transmisión de datos que se consigue es de 10Kbps, y puede ser utilizado tanto en viviendas ya construidas como de nueva construcción.

CEBus es un estándar muy ambicioso, y en él cooperan tanto la unión Europea como Japón, pero no existen muchos sistemas instalados, lo que se debe principalmente a la escasa oferta comercial de dispositivos y su elevado precio.

2.5.2 X10

El formato de codificación X-10 es un estándar usando transmisión de corrientes portadoras (*Power Line Carrier* = P.L.C). Se introdujo en el año 1978 para el Sistema de Control del Hogar de Sears y para los sistemas Plug´n Power de Radio Shack. Desde entonces, X-10 ha desarrollado y manufacturado versiones O.E.M (*Original Equipment Manufacturer*) de su Sistema de Control del Hogar para muchas compañías incluyendo Leviton Manufacturing Co., Stanley Healtth / Zenith Co., Honeywell, Norweb y Busch Jaeger, existiendo en la actualidad más de ocho millones de instalaciones. Todos estos sistemas utilizan el formato de codificación X-10 y son compatibles.

El sistema X-10 se caracteriza principalmente por:

- Ser un sistema descentralizado **configurable** pero **no programable**.
- De instalación sencilla (conectar y funcionar) y de fácil manejo por el usuario.
- Aplicable a instalaciones monofásicas y trifásicas.
- Gran madurez en el mercado con compatibilidad casi absoluta con los productos de la misma gama, obviando fabricante y antigüedad.
- Flexible y ampliable.

- Destinado a instalaciones pequeñas o medianas sin grandes prestaciones.

En la Figura 2-9 se presenta un ejemplo de instalación X10. La red de la instalación es la base de todo el sistema de corrientes portadoras (X-10). El elemento básico y fundamental de la técnica de corrientes portadoras es el aprovechamiento doble de la instalación eléctrica ya existente, como conductor de energía y de información. Con los componentes X-10 la red, además de suministro de corriente, se encarga también de la transmisión de señales de mando para los diversos aparatos eléctricos. Con ello se pueden enviar señales de corrientes portadoras a cualquier punto de la instalación que se desee, y a su vez pueden solicitarse de dicho punto las informaciones pertinentes.

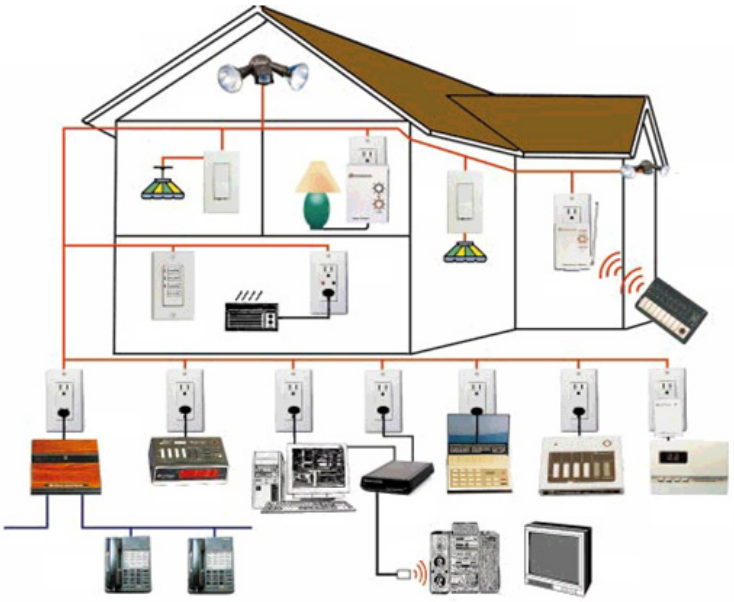


Figura 2-9. Ejemplo de una instalación X10.

El sistema permite el accionamiento a distancia y control remoto de diversos receptores eléctricos, desde uno o desde varios puntos y puede funcionar tanto en redes de corriente alterna monofásica como trifásica.

Principio de funcionamiento del protocolo X-10

Las transmisiones están sincronizadas con el paso por cero de la tensión de red, con la doble finalidad de, por una parte, sincronizar emisores y receptores y por otra, aprovechar el momento en que las interferencias producidas por los equipos eléctricos son mínimas. La codificación consiste en generar pulsos de 120KHz de 1ms de duración en el paso por cero (1 lógico: ausencia de pulso, 0 lógico: presencia de pulso).

Un mensaje completo en X-10 está compuesto por el código de comienzo (1110), seguido por la letra de dirección de casa y un código de control (véase la Figura 2-10).

Código de inicio + dir. de casa + código de control + sufijo

Figura 2-10. Mensaje de datos X10.

- La dirección de casa se codifica con cuatro bits, que permiten la identificación de 16 códigos diferentes, nombrados mediante letras de la A a la P. En los aparatos se selecciona mediante un conmutador rotativo.
- El código de control (4 bits) puede ser o una dirección de unidad o un código de comandos, dependiendo de si el mensaje es una dirección o un comando:
    - Los códigos de dirección se seleccionan mediante un segundo conmutador rotativo (números del 1 al 16).
    - Los códigos de control incluyen una serie de comandos definidos en el protocolo: encender, apagar, aumentar y reducir intensidad, etc.

El principio de codificación X10 permite una activación y respuesta definidas de hasta 256 receptores, puestos de control de aparatos o de grupos de consumidores. Con ello resulta posible el montaje de amplias redes.

Debido a las características del medio físico utilizado se transmite dos veces cada uno de estos bloques de información para que conseguir reducir las probabilidades de error en la transmisión. Además, cada par de bloques de información debe estar precedido por seis pasos por cero (tres ciclos de red), tiempo de espera es necesario para que el receptor procese los datos de dirección recibidos. En la Figura 2-11 se muestran los ciclos totales que necesita un transmisor para realizar una transmisión completa.

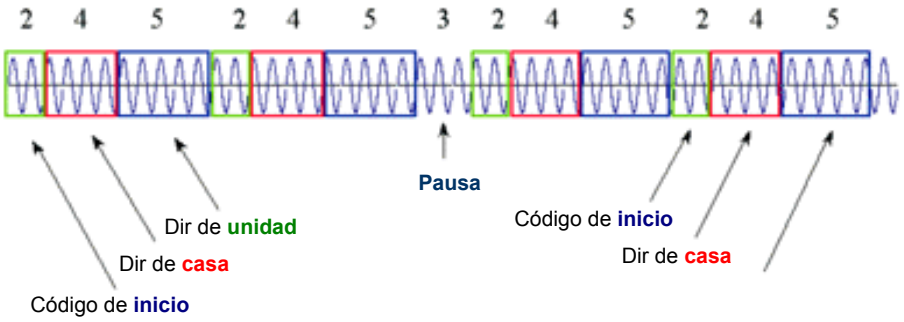


Figura 2-11. Ciclos para transmisión completa en X10.

Cada once ciclos de red se transmite un bloque de datos, y una transmisión estándar X-10 normal necesita 47 ciclos de la señal de red. A una frecuencia de 50 Hz esto supone un tiempo igual a **0,94 segundos en transmitir una orden completa**.

Además de la baja velocidad de transmisión, el protocolo no contempla mecanismos rigurosos de confirmación ni detección de errores, por lo que **la transmisión no es fiable**, y por tanto no se garantiza la correcta ejecución de las órdenes de control.

Dispositivos X10

Existen tres tipos de dispositivos X-10: los que sólo pueden transmitir órdenes, los que sólo recibirlas y los que pueden enviarlas y recibirlas. Los nodos están dotados de dos pequeños conmutadores giratorios, uno con 16 letras (A-P) y el otro con 16 números (1-

16), que permiten asignar una dirección de las 256 posibles. En una misma instalación puede haber varios receptores configurados con la misma dirección, todos realizarán la función preasignada cuando un transmisor envíe una trama con esa dirección. Cualquier dispositivo receptor puede recibir órdenes de diferentes transmisores.

También existe una serie de accesorios y componentes que ayudan a **solucionar problemas** en las instalaciones, entre los que podemos destacar los siguientes:

- **Acoplador / Repetidor - X10**. Asegura la calidad de la señal X10 cuando la distancia entre controlador y módulo receptor es demasiado larga y la señal sufre de atenuación. Además de amplificar la señal, la transmite en las tres fases por lo que serviría de acoplador en sistemas complejos no monofásicos.
- **Filtro Acoplador / Fases**. Este modulo X10 impide a las señales X10 sobre corriente portadora salir de la vivienda y ocasionar perturbaciones en otra instalación. Suprime las interferencias que vienen del exterior, como las parasitarias, órdenes X10 de otra instalación vecina. Además acopla las tres fases, en el caso de una instalación de corriente trifásica.
- **Programador / Verificador**. Es capaz de transmitir y recibir cada uno de los comandos, además de los comandos extendidos X10. Es una herramienta básica para instaladores de dispositivos X10: permite conocer niveles de ruido, niveles de señal, y otras.

En la Figura 2-12 se observa un ejemplo de instalación X10. En la cabecera de la red de baja tensión se colocan los filtros, acopladores y amplificadores de señal. Todos los dispositivos se conectarán a dicha red de 230V_{ac}. Los sensores, tales como pulsadores, o entradas binarias para detectores serán los encargados de enviar órdenes, y los actuadores (módulos de pared, empotrados o de enchufe) activarán las cargas en función de los comandos recibidos. También pueden existir controladores más complejos (como un ordenador) para funciones implementar servicios más complejos.

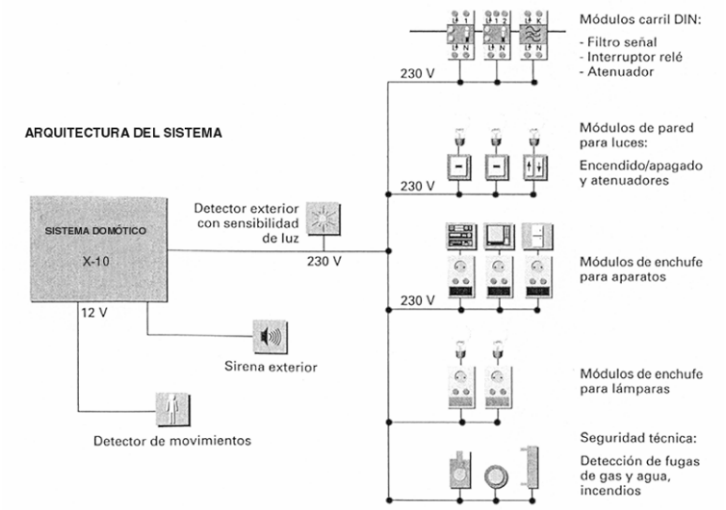


Figura 2-12. Componentes en un sistema X10 [X10 08].

2.5.3 Lonworks

La compañía Echelon, responsable del desarrollo de esta tecnología, surgió como una iniciativa de Mike Markkula (exdirectivo de Fairchild Semiconductor, Intel y Apple), que lo propuso en el año 1990. Inicialmente se pretendía ocupar el rango de aplicaciones que X-10, por su simplicidad, no alcanzaba a cubrir, pero actualmente el ámbito de aplicación de este sistema abarca desde industrias, edificios, viviendas y automóviles hasta cualquier otro pequeño dispositivo susceptible de ser controlado [Byoug 00].

El protocolo de comunicación empleado, **LonTalk**, es un protocolo abierto (previo pago de tasas) de comunicaciones basado en el modelo de referencia OSI de ISO.

Los componentes básicos de una red LonWorks son dos:

- Neuronas. Son unos circuitos integrados que contienen dispositivos de entrada/salida, tres microprocesadores y memoria en la que reside el sistema operativo.
- Transceptores. Son dispositivos emisores-receptores que se encargan de conectar las neuronas con el medio de transmisión.

Existe también un sistema de desarrollo, LonBuilder, que consiste en un software y dos emuladores de neuronas que pueden comunicarse entre sí. Las neuronas ('*neuron chips*'), fabricadas por Toshiba y Motorola, constituyen el nodo básico de las redes de control. Mediante los transceptores se consigue que el protocolo de comunicación sea totalmente independiente del medio de transmisión utilizado y con la herramienta *LonBuilder* se pueden desarrollar aplicaciones orientadas a redes.

Los medios de transmisión disponibles son cinco:

- Par trenzado (categoría IV) de cinco hilos: dos de datos, dos de alimentación y uno de tierra.
- Fibra óptica.
- Línea de baja tensión.
- Radiofrecuencia.
- Cable coaxial.

El protocolo de ese sistema implementa todos los niveles del modelo de referencia OSI, como se ilustra en la Tabla 2-4. En cuanto a la topología del cableado de la red, existe versatilidad para emplear cualquiera de las existentes (véase la Figura 2-13). La topología en bus requiere de dos elementos de terminación en ambos extremos para su buen funcionamiento (R_l=105 Ω). Se suele utilizar en aplicaciones industriales con fibra óptica o par trenzado.

Las topología libre y en anillo (R_l=52.3 Ω), tan sólo necesitan de una terminación que se puede colocar en cualquier lugar. En la Tabla 2-5 se resumen las características de transmisión sobre par trenzado en función de la topología.

Nivel	Características principales
1. Físico	Puede utilizar: PL, TP, IR, RF, CX y/o FO.
2. Enlace	CSMA/CA (con prioridad opcional) y CMA/CD. La codificación es Manchester diferencial.
3. Red	Emisión de ACK y UNACK. Transmisión uni/multi-difusión. Servicios de direccionamiento, etc.
4. Transporte	*Servicios de mensajes* hacia el exterior, desde el exterior, detección de duplicidades, posibilidad de autenticación, etc. *Servicio de transportes* tanto unidifusión y de difusión, repetición de UNACK, etc.
5. Sesión	Pregunta – respuesta.
6/7. Presentación/Aplicación	Propagación de variables de redes, mensajes genéricos de paso, mensajes de gestión de la red, mensajes de diagnósticos de la red, transmisión de tramas externas, etc.

Tabla 2-4. Protocolos implementados en Lonworks y equivalente OSI.

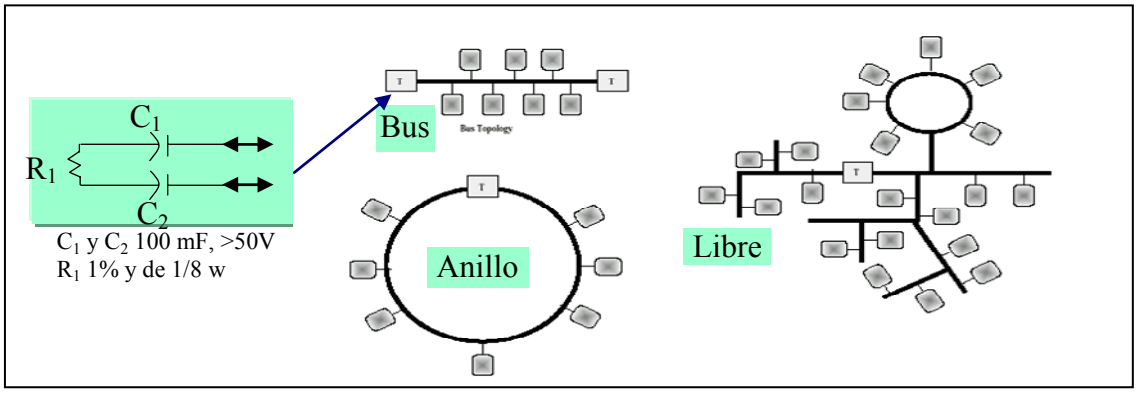


Figura 2-13. Topologías utilizables en Lonworks.

Longitud del bus	Velocidad de transmisión	Nº máximo de nodos
Bus con terminación doble		
130 m	1,25 Mbps	64
2700 m	78kbps	64
Topología libre		
500 m	78 kbps	128

Tabla 2-5. Características de la transmisión sobre par trenzado.

Existen varios tipos de **direcciones:**

- *Dirección física (Neuron ID)*: consiste en una dirección de 48 bits que viene grabada de fábrica (como la dirección MAC en una tarjeta de red).
- *Dirección de dispositivo:* está formada por tres campos:

- *ID de dominio (domain ID):* un dominio es una colección de dispositivos que pueden interoperar (en una red virtual), localizados en uno o más canales. Se pueden tener hasta 32.385 dispositivos en un dominio. Ocupa 6 bytes.
- *ID de subred (subnet ID)*: abarca hasta 127 dispositivos en un canal o canales conectados por repetidores (mismo enlace de datos). Se utilizan para soportar encaminamiento eficiente en redes grandes. Puede haber un máximo de 255 subredes dentro de un dominio.
- *ID dispositivo (node ID):* identifica al dispositivo en la subred.

- *Dirección de grupo:* colección lógica de dispositivos en un dominio (no importa su situación física). Se pueden agrupar hasta 63 nodos. No puede haber más de 256 grupos en un dominio.

Un nodo puede pertenecer como máximo a 2 dominios. Cada nodo tiene una dirección de subred y una dirección de nodo para cada dominio al que pertenezca. Asimismo, un nodo puede pertenecer a 15 grupos como máximo en cualquier dominio en el que esté.

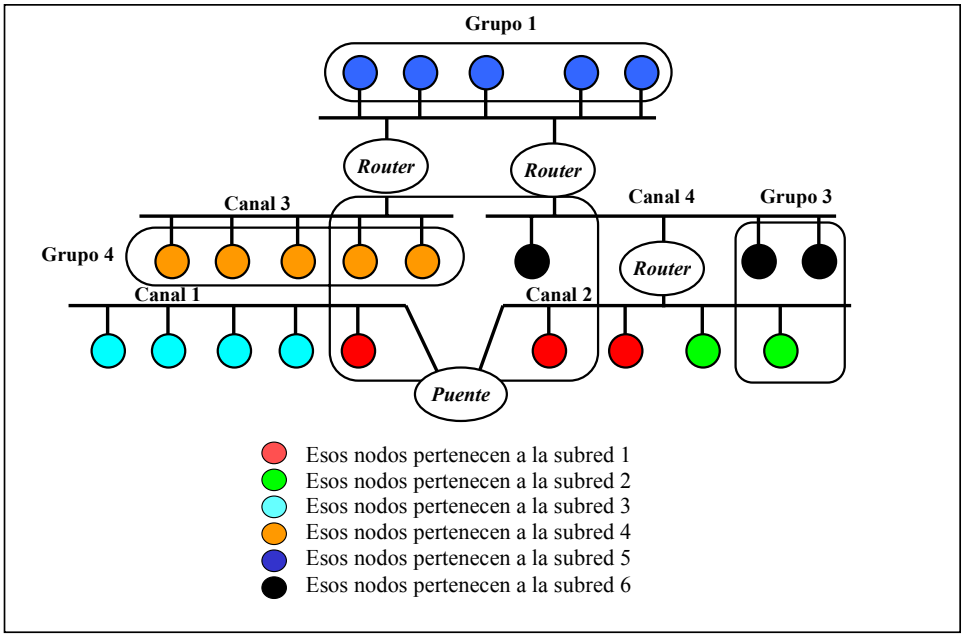


Figura 2-14. Dominio LonTalk.

También se utilizan direcciones de difusión en un dominio o una subred (a veces se utilizan en lugar de las de grupo para preservarlas). En la Figura 2-14 se muestra un ejemplo de la topología: un canal es la unión física de distintos nodos y puede estar formado por nodos que pertenezcan a distintas subredes. Un grupo es la unión lógica de distintos nodos y puede estar formado por miembros de distintas subredes y canales. Es decir, un grupo no depende de la topología ni del medio físico que se emplee. Una única red puede abarcar distintos canales mediante puentes. Se utilizan varios dominios cuando se excede el número máximo de dispositivos o se quieren separar para que no puedan interoperar.

El **formato de las tramas** (véase la Figura 2-15) está constituido por un campo de control, la dirección de nodo, la dirección de dominio, los datos de usuario y un campo de CRC (código de redundancia cíclica). El tamaño máximo del campo de datos es de 228 bytes.

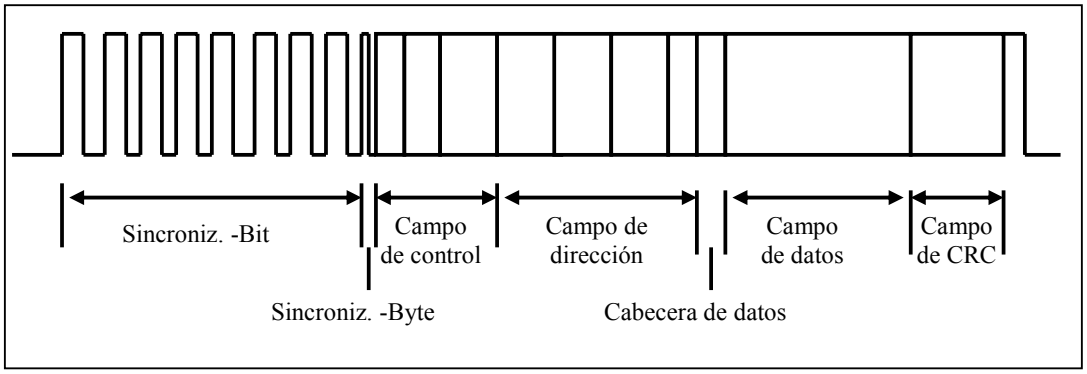


Figura 2-15. Formato de la tramas Lonworks.

El proceso de instalación de una red Lonwork se realizara en tres fases:

Direccionamiento. Cada nodo tiene un identificador (*ID number*) de 48 bits que viene de fábrica. Se conecta un ordenador personal con el software de control de la red a través del puerto serie, y con él se obtiene este identificador mediante la pulsación del botón de servicio del nodo. Una vez configurado este número, el programa le proporciona una nueva dirección de red (dominio + subred + nodo), que queda almacenada en su memoria RAM.

Establecimiento de enlaces lógicos de relación entre nodos. Con este proceso se asigna a cada nodo la dirección o direcciones a las que va a mandar sus mensajes.

Configuración de cada uno de los nodos, con lo que se completa la instalación lógica de éstos. Cada nodo suele tener un conjunto de parámetros que han de ser configurados por el instalador, como por ejemplo, velocidad de la transmisión, márgenes de alarma, comportamiento de los sensores o actuadores conectados (normalmente abierto o cerrado, envío de comandos de ON, OFF o alternados, etc.). Los procesos de configuración y de enlace lógico son básicos para la implementación de la funcionalidad de la red domótica. Este proceso de enlace, denominado ***binding*** (véase la Figura 2-16), consiste en asociar ***variables de red*** de los sensores con *variables de red* de los actuadores. Una vez asociadas las variables de red, los nodos intercambian información a través de la red para la ejecución de los comandos.

Los tipos de variables de red están normalizados en el protocolo para garantizar la compatibilidad entre dispositivos. Existen tipos para todos los comandos ejecutables en la red, como encendido/apagado, subir/bajar luminosidad, valores absolutos, valores en punto flotante, etc.

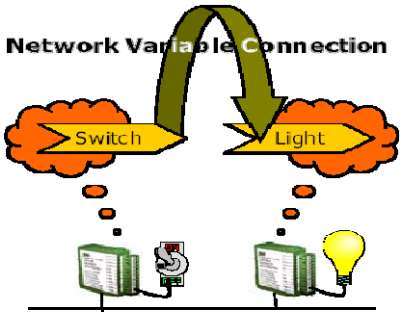


Figura 2-16. Asociación de variables de red.

Todo este proceso de direccionamiento, configuración y asociaciones lógicas se realiza mediante la aplicación *Lonmaker*, que se ejecuta en un ordenador conectado a la red Lon a través de un transceptor al medio físico utilizado. Existen transceptores con conectividad (en el lado del PC) serie, USB, IP, etc.

2.5.4 EHS

A finales de los años 80 la Unión Europea propició el desarrollo de un par de proyectos SPRIT (el *Home System 2341* y el *Integrated Interactive Home Project*), de los que surgiría la *European Home System Association* (EHSA) en 1990, de la que inicialmente formaban parte compañías como ABB, BT, Legrand, Philips, Siemens, Thomson y Thorn EMI.

Los objetivos de esta asociación fueron:

- Posibilidad de interoperación entre los distintos equipos de diferentes fabricantes.
- Fácil instalación y reconfiguración por parte del usuario.
- Posibilidad de integración de todos los dispositivos y medios disponibles en una vivienda convencional.

El bus EHS surgió como un sistema abierto, consecuencia de esta iniciativa, con control y gestión distribuida, y preparado para su uso en distintos medios simultáneamente. Sigue el modelo de referencia OSI, implementando únicamente las capas física, de enlace, de red y de aplicación (véase la Figura 2-17).

Los medios físicos que se pueden emplear son: red eléctrica (PL), par trenzado de clases 1 y 2 (TP1 y TP2), cable coaxial, radio frecuencia e infrarrojos (véase la Figura 2-17). Todos los medios pueden distribuir señales de clase 1 (señales de control), algunos distribuyen además señales de clase 2 (voz/datos baja velocidad) e incluso señales de clase 3 (audio/video/datos alta velocidad). Algunos medios también pueden distribuir la alimentación de los dispositivos.

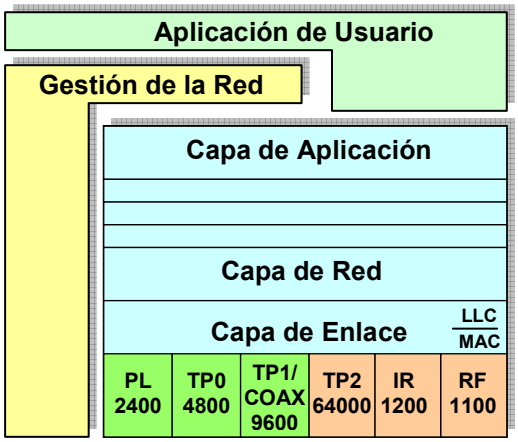


Figura 2-17. Capas del modelo OSI implementadas en EHS.

Tipo de medio	TP1	TP2	CX	PL	RF	IR
Uso	Propósito general Control	Telefonía, RDSI, Datos y/o control	AV, TV Datos y/o control	Control	Telefonía Control	Control remoto
Velocidad	9,6 Kbps	64 Kbps	9,6 Kbps	2,4 Kbps	1,1 Kbps	1,2 Kbps
Protocolo MAC	CSMA/CA	CSMA/CD	CSMA/CA	CSMA/ack	CT2	-
Alimentación	35 V	35 V	15 V	230 V_{ac}	-	-
Canales de información	-	14	Muchos	-	40	-
Velocidad	-	64 Kbps	Analógica	-	32 Kbps	-
Codificación	-	TDM	FDM	-	FDM	-
Topología	Libre	Bus	Bus	Libre	Libre	Libre
Nº máximo de nodos	128	40	128	256	256	256
Rango	500 m	300 m	150/50 m	Casa	50-200 m	Habitación

Tabla 2-6. Características de los diferentes medios de transmisión en EHS.

En EHS se pueden implementar tantas aplicaciones como dispositivos y funcionalidades se encuentren en un hogar. Cada dispositivo está asociado a una determinada área de aplicación, dentro de la cual el elemento es un objeto. Para definir cada objeto se utilizan dos bytes, uno para el área (*application area*), y otro para el dispositivo (*device descriptor*).

Existen diversas **áreas de aplicación**, como telecomunicaciones y audio/vídeo, electrodomésticos, calefacción, iluminación, etc. Los dispositivos EHS pueden ser de seis tipos (véase la Figura 2-18):

- **Dispositivos simples** (SD: *simple devices*). Tienen funcionalidad autónoma propia, pero no son capaces de gestionar autónomamente la integración en un sistema (por ejemplo actuadores on/off, etc.).
- **Dispositivos complejos** (CoD: *complex devices*). Son como los anteriores pero sí tienen capacidad para integrarse autónomamente al sistema.
- **Encaminadores** (*routers*). Permiten la interconexión de distintos medios en EHS.
- **Pasarelas** (*Gateways*). Integran distintos sistemas.
- **Coordinador de dispositivos** (DC: *device coordinator*). Sirven de pasarela entre los dispositivos simples y los controladores de prestaciones (FC). No tienen funcionalidad autónoma propia, pero son capaces de gestionar de modo autónomo la integración en un sistema de dispositivos simples.
- **Controlador de prestaciones** (FC: *feature controller*). Utilizan las prestaciones de los dispositivos simples (a través de los coordinadores) y complejos. Proporcionan inteligencia a la aplicación en el sentido de control, monitorización, toma de decisiones, etc.

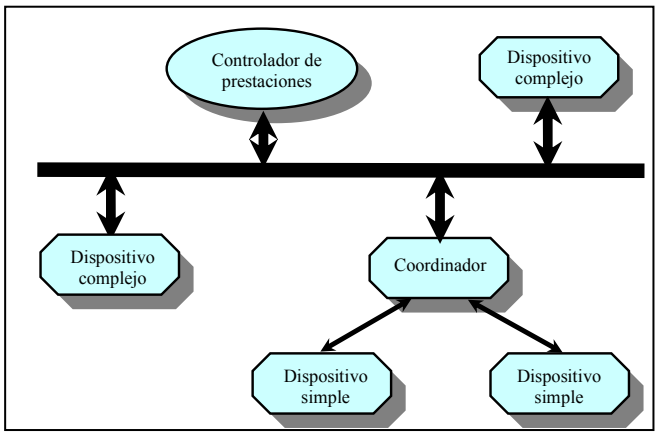


Figura 2-18. Esquema de comunicación entre elementos EHS.

Una red EHS puede estar formada por distintas subredes EHS, e incluso por redes distintas a EHS, en cuyo caso se emplean pasarelas (véase la Figura 2-19).

En EHS cada dispositivo recibe el nombre de unidad. Cada unidad conectada a una subred tiene su propia dirección de subred. Una dirección de unidad se compone de la dirección de subred de la unidad destinataria, el número de rutas y las direcciones de los distintos encaminadores para alcanzar la subred de destino. La dirección de subred se puede definir en el nivel de aplicación bien mediante mini-interruptores existentes en cada dispositivo, bien mediante un procedimiento de registro.

El **procedimiento de registro** es una función de EHS que permite la asignación dinámica de direcciones. Por ejemplo, si dos unidades de dos subredes de pares trenzados tienen la misma dirección, al mover una de las unidades a la otra subred, habría un problema, que se soluciona con el procedimiento de registro (*registration procedure*). Este procedimiento tiene lugar en el momento de la instalación (registro de

categoría I) o cada vez que el sistema se pone en funcionamiento (registro de categoría II).

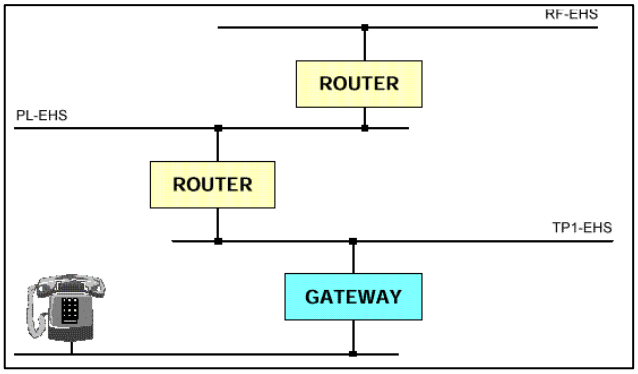


Figura 2-19. Integración de distintas subredes en una red EHS.

Mediante este procedimiento, cada unidad nueva conectada a la red 'negocia' su dirección a través de una unidad denominada Controlador de Medios (MdC), que es la responsable de la asignación de direcciones en cada subred. La unidad MdC es opcional, ya que sus tareas pueden ser realizadas por un controlador de prestaciones (FC).

Cuando no hay un MdC en la subred, el registro se hace mediante un mecanismo de asignación distribuida de direcciones (DAA). Las acciones llevadas a cabo en este registro son:

- La unidad elige una dirección de modo aleatorio y manda un mensaje a esa dirección.
- Si no recibe respuesta, la unidad mantiene esa dirección.
- Si hay respuesta, la unidad elige una nueva dirección y repite el proceso hasta que obtenga una dirección propia.

Para la cooperación de las diferentes unidades dentro de una aplicación deben crearse una serie de vínculos entre ellas. Esto es lo que se conoce como **procedimiento de enrolado**. Este procedimiento requiere que las unidades intercambien sus direcciones, y es esencial para el funcionamiento autónomo del sistema, ya que permite a las unidades detectar la presencia de las demás.

El enrolado comienza al encender una unidad, una vez completado el registro, y se realiza llevando a cabo las siguientes acciones:

- Un controlador de prestaciones (FC) difunde su petición de descriptores de dispositivo (DD) a todos los dispositivos complejos (DoC). Este mensaje utiliza una dirección de grupo predeterminada para alcanzar a todos los dispositivos complejos (CoD's).
- Los dispositivos complejos (CoD's) reciben el mensaje junto con información adicional que les permite conocer la dirección de su controlador de prestaciones

(FC). Los CoD's envían entonces su descriptor de dispositivo (DD) al controlador de prestaciones (FC), usando su propia dirección.

- El controlador de prestaciones (FC) recibe los descriptores de dispositivo (DD) de los distintos dispositivos complejos (CoD's) junto con sus direcciones. Si el FC estuviera interesado en un CoD concreto, enviaría su mensaje de enrolado positivo al CoD en cuestión.
- El controlador de prestaciones (FC) y el dispositivo complejo (CoD) quedan ya enrolados y cada uno almacena la dirección individual del otro dispositivo.

La **estructura de la trama** EHS se compone de los siguientes campos (véase la Figura 2-20):

- Preámbulo (en PL) para sincronización del envío de datos entre los dispositivos emisor y receptor.
- Cabecera, que marca el inicio de los datos y permite reconocer una trama EHS.
- La dirección de vivienda permite discriminar si una trama viene de otra casa.
- Código de prioridad para definir el nivel de prioridad del mensaje.
- Direcciones de los dispositivos de origen y destino.
- Datos, con los datos de útiles del mensaje (información de la acción de control a realizar o datos a transferir).
- Campo de corrección de errores, en el que se utilizan 2 bytes para garantizar la fiabilidad de la comunicación.

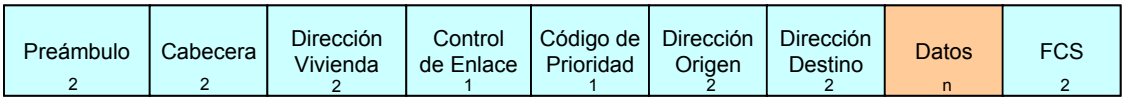


Figura 2-20. Estructura de las tramas EHS.

2.5.5 KNX/EIB

El Bus de Instalación Europeo (EIB, *European Installation Bus*) surgió con la idea de introducir en el mercado un sistema unificado para la gestión de edificios, creado por el consorcio europeo EIBA (*European Installation Bus Association*) en 1990 por más de setenta compañías (ABB, Siemens, Jung, etc.).

En la actualidad la asociación tiene más de cien miembros, existiendo unas veinte empresas que suministran productos, siendo las más importantes Siemens, ABB, Temper, Grasslin y Niessen. También existen miembros científicos que colaboran en el desarrollo de actividades de I+D, especialmente universidades y centros de investigación.

Las funciones de la asociación son básicamente el soporte para la preparación de normas unificadas y la definición de las pruebas y requisitos de homologación que garanticen la calidad y compatibilidad de los productos. En la actualidad, EIBA se ha integrado dentro de la asociación Konnex (KNX) para la convergencia de sistemas europeos.

Se trata, además, de un sistema abierto sobre las mismas premisas que otros sistemas de comunicación como los buses de campo abiertos: tanto las especificaciones del protocolo como los procedimientos de verificación y certificación están disponibles, así como los componentes críticos del sistema (microprocesadores específicos con la pila del protocolo y electrónica de acoplamiento al bus).

KNX/EIB está recogido en normas europeas e internacionales (EN 50090 e ISO/IEC 14543-3-X), y es la tecnología domótica abierta más instalada en España y a nivel europeo [Mint 08]. Por este motivo, se va a trabajar con la tecnología KNX/EIB como demostrador de la metodología que se propone más adelante en este Trabajo de Tesis.

2.5.5.1 Tecnología

El KNX/EIB es un sistema distribuido (no requiere de un controlador central de la instalación), en el que todos los dispositivos que se conectan al bus de comunicación de datos tienen su propio microprocesador y electrónica de acceso al medio.

Existen diferentes medios físicos para la interconexión de dispositivos: cable de par trenzado dedicado y red eléctrica de baja tensión (modos TP1 y PL110 del estándar KNX, que se describe en el apartado "2.5.6 Convergencia de Sistemas") y radio-frecuencia. La diferencia entre los dispositivos de los tres tipos radica en la electrónica de acceso al medio, siendo el resto del protocolo de comunicaciones común a todos ellos.

La instalación sobre red eléctrica de baja tensión, que funciona por corrientes portadoras de manera similar a otros sistemas, como X10, se reserva a viviendas o edificios ya construidos, donde la instalación de nuevo cableado sería muy costosa. No obstante, este tipo de medio es muy poco empleado por mayor coste y menor fiabilidad. Por ello, se van a describir las características con referencia al medio más utilizado: cable de bus dedicado de tipo par trenzado (KNX/TP1).

Los sensores son los responsables de detectar cambios de actividad en el sistema (operación de un interruptor, movimientos, cambio de luminosidad, temperatura, humedad, etc.), y ante éstos, transmitir mensajes (denominados telegramas) a los actuadores, que se encargan de ejecutar los comandos adecuados. Los sensores funcionarán por tanto como entradas al sistema, y los actuadores como salidas para la activación y regulación de cargas.

En la versión de par trenzado (KNX/TP1), la línea de bus, que sirve como soporte para la transmisión de datos y alimentación de los nodos, llega a todos los dispositivos, pero la red eléctrica sólo se conectará a los elementos actuadores para el control de las cargas (iluminación, motores de persianas, etc.) (véase la Figura 2-21).

Los datos se transmiten como una tensión alterna superpuesta sobre la alimentación en corriente continua del bus, empleando para ello únicamente dos hilos. Para ello es necesario cumplir dos requisitos:

- Cada línea dispone de una fuente de alimentación, con un filtro en serie con el bus (bobina) para evitar el filtrado de los datos de alta frecuencia.
- Hay que desacoplar los datos de la componente de alimentación continua en cada dispositivo.

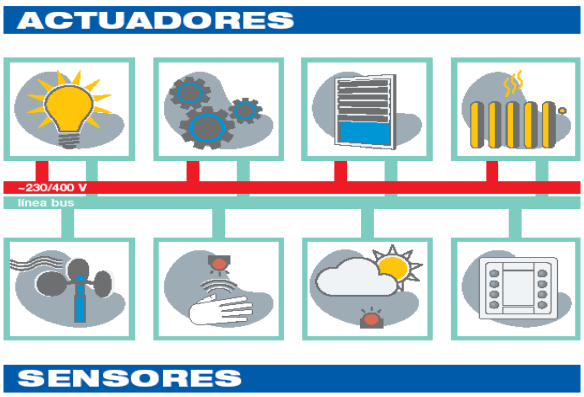


Figura 2-21. Esquema general de una instalación KNX/EIB.

La codificación es muy sencilla (véase la Figura 2-22), el 1 lógico se codifica como un impulso simétrico de ±5V sobre la tensión en el bus, y el 0 lógico como ausencia de dicho impulso. En la práctica, el generador del nodo sólo genera el impulso negativo, y la parte positiva se obtiene como consecuencia de la fuerza contraelectromotriz de la bobina de filtrado existente en la fuente de alimentación de la línea.

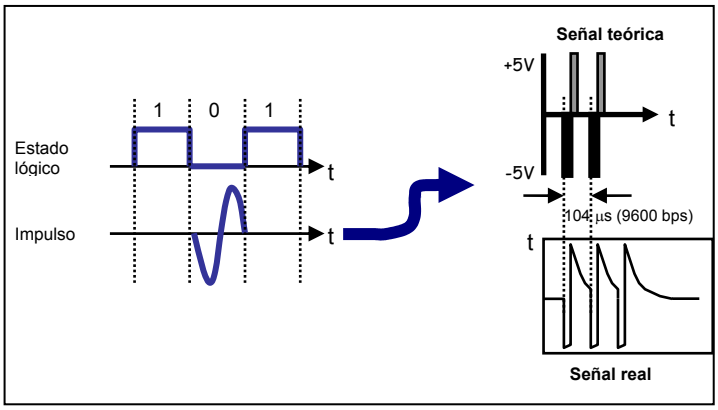


Figura 2-22. Codificación de datos en KNX/EIB.

Esta codificación permite utilizar un método de acceso al medio denominado CSMA/CA[1], en el que las colisiones se resuelven mediante un campo de prioridad en la cabecera de los mensajes. Las transmisiones se realizan en modo asíncrono a 9600bps.

[1] CSMA/CA: *Carrier sense multiple access / Collision avoidance*. Acceso múltiple por detección de portadora, evitando colisiones.

2.5.5.2 Topología

Para el conexionado de dispositivos del bus en cada línea se permite cualquier topología: árbol, estrella, bus o anillo, lo que facilita la **escalabilidad** de la instalación en viviendas y edificios de cualquier tamaño. La topología de conexión de dispositivos contempla tres niveles de conexionado, como se muestra en la Figura 2-23.

La **línea** es la unidad mínima de instalación. En ella se pueden conectar hasta 64 dispositivos (dependiendo de la capacidad de la fuente de alimentación y de la carga máxima producida por los dispositivos existentes). Si se desean conectar más componentes al bus, habrá que de instalar una nueva línea, que se acoplará, junto con la primera, a una línea principal mediante acopladores de línea. Se pueden acoplar hasta 15 líneas en la línea principal, constituyendo un **área**. De este modo, en un área se pueden conectar hasta 960 dispositivos.

Cabe la posibilidad de unir hasta un total de 15 áreas distintas mediante los denominados Acopladores de Área para constituir el **sistema completo** (véase la Figura 2-23), que permitiría integrar hasta un máximo de 14.400 dispositivos.

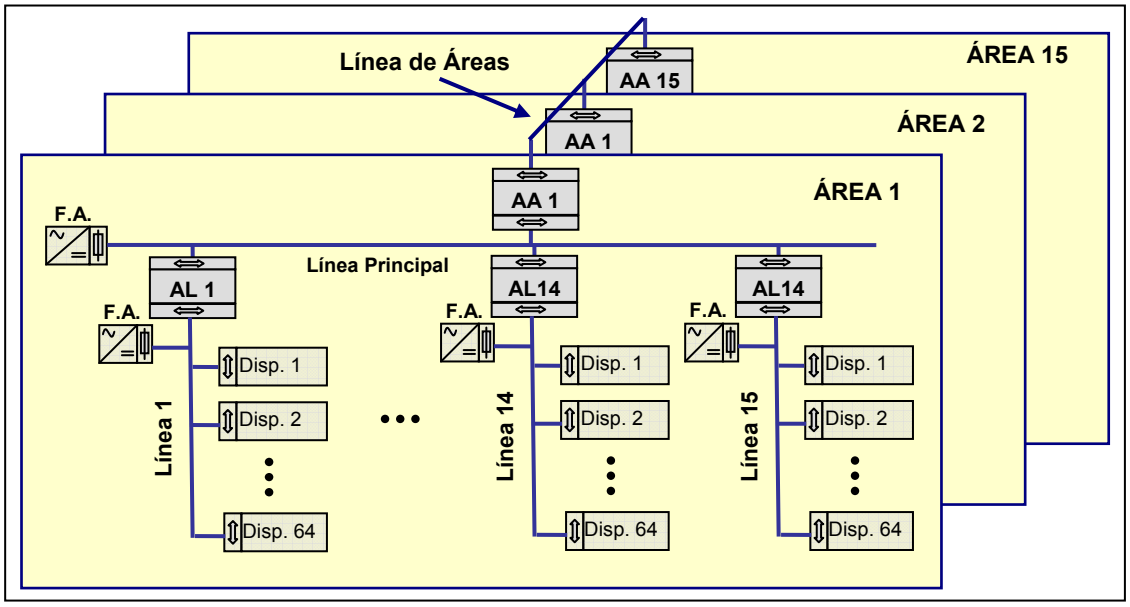


Figura 2-23. Arquitectura de una red KNX/EIB.

Cada línea, tanto la principal como las secundarias, deben tener su propia fuente de alimentación. Además, la línea principal y la línea de áreas pueden tener conectados directamente hasta 64 dispositivos (incluyendo los acopladores de línea o área).

Los acopladores de línea y área son dispositivos de encaminamiento que se encargan de separar físicamente las líneas (disponen de doble electrónica de acoplamiento) y de filtrar los mensajes que se intercambian.

2.5.5.3 Direccionamiento

Los diferentes elementos existentes en una instalación KNX/EIB quedan perfectamente identificados gracias al sistema de **direccionamiento**. Existen dos tipos de direcciones: direcciones físicas y direcciones de grupo.

Las **direcciones físicas** identifican unívocamente cada dispositivo y se corresponden con su localización en la topología global del sistema (área – línea secundaria – dispositivo). La dirección física consta de tres campos, que se representan separados por puntos, por ejemplo 1.1.3, identificando a un dispositivo situado en el área 1, línea 1, con dirección 3. En la Figura 2-24 se muestra un ejemplo de direcciones físicas asignadas a los dispositivos de un sistema EIB:

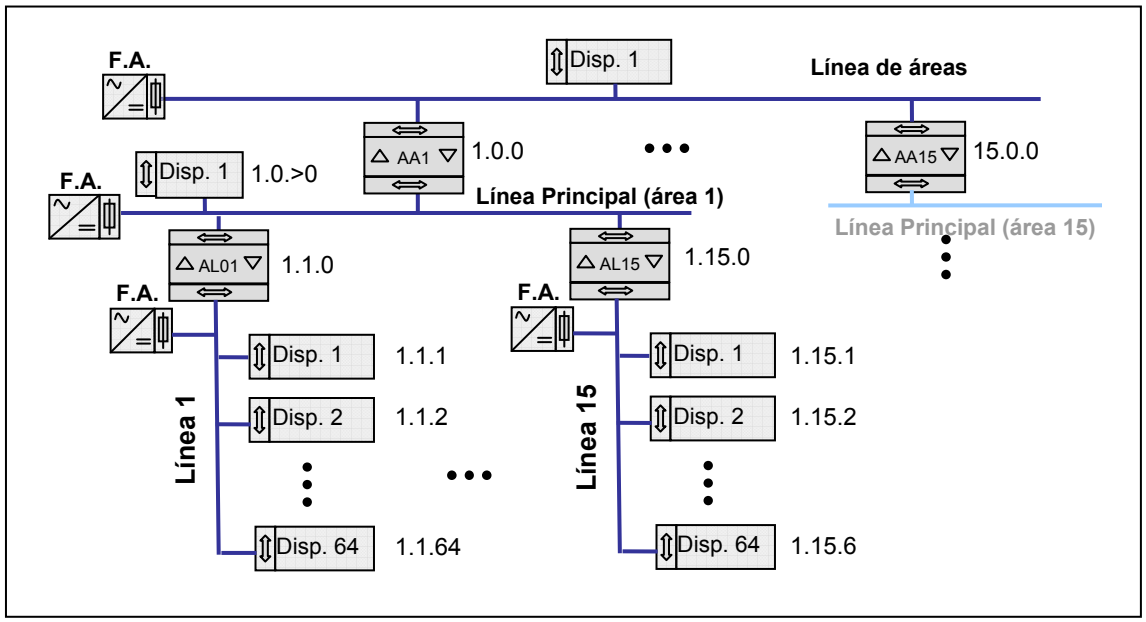


Figura 2-24. Ejemplo de direccionamiento de dispositivos KNX/EIB.

Las **direcciones de grupo** se emplean para definir funciones específicas del sistema, y son las que determinan las asociaciones de dispositivos en funcionamiento (y la comunicación entre sus objetos de aplicación). Se trata de un campo de 15 bits con el formato descrito en la Figura 2-25. Su asignación es libre en el diseño de la instalación y los distintos niveles se suelen utilizar para organizarlas por funciones (por ejemplo, grupo principal para identificar plantas del edificio, grupo medio para funciones como iluminación, persianas, etc., y subgrupo para la función concreta "conmutar luz salón").

En KNX/EIB los nodos de la red disponen de un programa de aplicación en el que se definen una serie de parámetros configurables y los **objetos de** comunicación (véase la Figura 2-26). Estos objetos son entidades abstractas que tienen correspondencia con los recursos hardware del dispositivo.

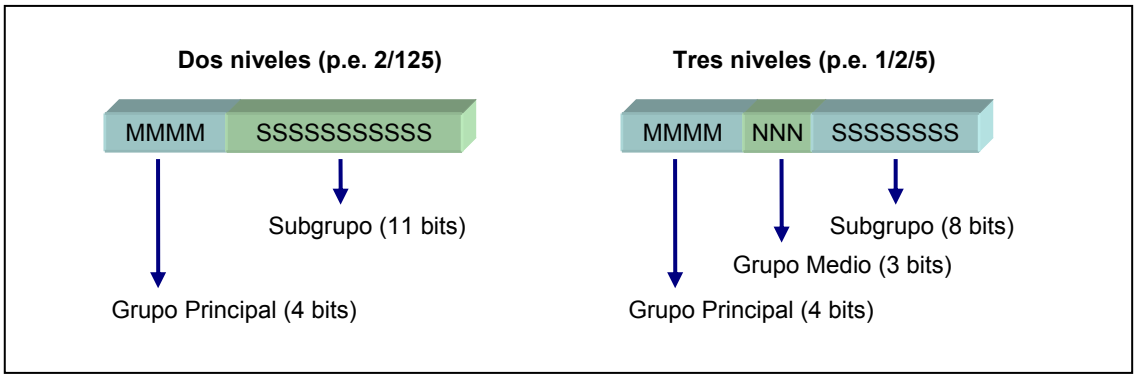


Figura 2-25. Direcciones de grupo en KNX/EIB.

Este concepto es similar al descrito como enlace lógico de variables de red en la tecnología Lonworks, que equivale a la asignación de direcciones de grupo a los objetos de comunicación en KNX/EIB.

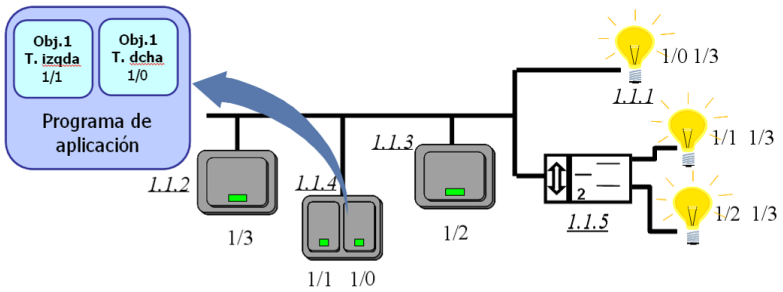


Figura 2-26. Ejemplo de asignación de direcciones y de objetos de aplicación en KNX/EIB.

2.5.5.4 Formato de las transmisiones

El envío de un mensaje o telegrama se realiza cuando se produce un evento, por ejemplo, la activación de un pulsador o la detección de presencia, y en algunos casos de forma cíclica (por ejemplo, en órdenes de regulación de temperatura desde un termostato). El envío se realiza utilizando el mecanismo de acceso al medio CSMA/CA descrito con anterioridad, y los destinatarios responden con reconocimiento (*Ack*). Si la recepción es incorrecta, no se recibe reconocimiento (o bien se recibe no reconocimiento), la transmisión se realiza un número programado de reintentos.

El mensaje que se transmite por el bus, denominado telegrama, contiene la información específica sobre el evento que se ha producido. Tiene siete campos, seis de control para conseguir una transmisión fiable y un campo de datos útiles con el comando a ejecutar (véase la Figura 2-27).

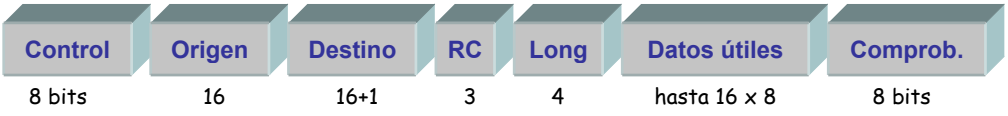


Figura 2-27. Formato de los telegramas KNX/EIB.

De este modo, a 9600bps, la transmisión de un byte supone un tiempo de 1.35 ms, y la de un telegrama completo entre 20 y 40 ms (la mayoría de las órdenes son de marcha-paro y suponen un tiempo de envío de 20 ms).

Durante el funcionamiento normal de la instalación, en el campo de destino se incluye la dirección de grupo, lo que permite el envío de órdenes de control entre los sensores y actuadores de la instalación. Estas órdenes o comandos se incluyen en el campo de datos útiles y tienen un formato definido en el estándar (al igual que ocurre con las variables de red en la tecnología Lonworks). El *EIB Interworking Standard (EIS)* recoge los tipos de datos normalizados junto con su tamaño y función[2], tal como se puede apreciar en la Tabla 2-7.

Nº EIS	Función EIB	Nº bytes	Descripción
EIS 1	Conmutación (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	Día de la semana, hora, minutos y segundos.
EIS 4	Fecha (date)	3 bytes	Día/mes/año (el margen es de 1990 a 2089).
EIS 5	Valor (value)	2 bytes	Para enviar valores físicos con representación S,EEEE,MMMMMMMMMM.
EIS 6	Escala (scaling)	8 bit	Se utiliza para transmitir valores relativos con una resolución de 8 bit. Por ejemplo, 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 bit (tanto con signo como sin signo).
EIS 15	Cadena (Character String)	14 bytes	Transmite un cadena de caracteres ASCII de hasta 14 bytes.

Tabla 2-7. Tipos EIS (EIB Interworking Standard).

El EIS contiene los datos útiles para cada función asignada a los objetos de comunicación. Según este estándar existen siete tipos diferentes, cada uno asignado a un tipo de acción de control (conmutación, regulación de luz, envío de valor absoluto, envío de valor en punto flotante, etc.). De este modo se garantiza la compatibilidad entre dispositivos del mismo tipo de diferentes fabricantes.

[2] En Konnex los EIS se denominan DPT (*Data Point Types*), pero conservan la misma estructura que los EIS.

2.5.5.5 Programación de los dispositivos

La programación supone la etapa final de la realización de un proyecto de instalación KNX/EIB. Se realiza habitualmente conectando un ordenador personal al bus mediante una pasarela (RS232, USB o IP) y requiere realizar las siguientes tareas:

- Programar las **direcciones físicas** de los dispositivos de acuerdo con la topología utilizada
- Cargar en el software de diseño (ETS: *Engineering Tool Software*) los **programas de aplicación** que se van a utilizar para los dispositivos. Estos programas de aplicación son proporcionados por los fabricantes en forma de librerías.
- Configurar una serie de parámetros que definen el funcionamiento del programa de aplicación.
- Realizar los enlaces lógicos entre sensores, actuadores y controladores mediante la asignación de direcciones de grupo a los objetos de comunicación de los dispositivos.
- Descargar los programas de aplicación con los parámetros y asociaciones definidas en los dispositivos de la red.

2.5.6 Convergencia de Sistemas

En apartados anteriores se ha aportado una panorámica de las características y situación de los sistemas domóticos más importantes existentes en la actualidad. Habitualmente, la gestión de sistemas en edificios y viviendas se ha basado en soluciones que muchas veces son específicas y propietarias, lo que dificulta en gran medida el integrar todos los servicios de gestión y mantenimiento, que es en definitiva el objetivo de la domótica. Esto, unido a las diferencias existentes en las especificaciones de los distintos sistemas, ha hecho que ninguno de ellos haya alcanzado la aceptación necesaria para imponerse en este sector.

Por lo tanto, se hacen necesarias soluciones abiertas, que garanticen la integración de dispositivos y sistemas de diferentes fabricantes, permitiendo a su vez un elevado grado de flexibilidad en caso de extensiones o cambios en las especificaciones del sistema. A tal efecto, y para conseguir un estándar **común** y **abierto**, siguiendo las directrices de la CE para HBES, en 1999 las tres asociaciones de mayor peso europeo se fusionaron en una asociación común (Konnex):

- BCI (*Batibus Club International*), responsable del sistema Batibus.
- EIBA (*European Installation Bus Association*), del sistema EIB.
- EHSA (*European Home Systems Association*), del sistema EHS.

El marco de la arquitectura abstracta que constituye este estándar común, también denominado **modelo de convergencia KNX,** está basado en el modelo de referencia OSI, definido por la ISO. Los objetivos de esta iniciativa son:

- Crear un único estándar para la domótica e inmótica que cubra todas las necesidades y requisitos de las instalaciones profesionales y residenciales de ámbito europeo.
- Aumentar la presencia de estos buses domóticos en áreas como la calefacción, ventilación y aire acondicionado (HVAC).
- Mejorar las prestaciones de los diversos medios físicos de comunicación, sobre todo en la tecnología de radiofrecuencia.
- Introducir nuevos modos de funcionamiento que permitan aplicar una filosofía *Plug&Play* a muchos de dispositivos típicos de una vivienda.
- Contactar con empresas proveedoras de servicios (eléctricas y de telecomunicaciones) con el objeto de potenciar las instalaciones domóticas.

En resumen, se trata de, partiendo de los sistemas EIB, EHS y BatiBUS, crear un único estándar europeo que sea capaz de competir en calidad, prestaciones y precios con otros sistemas norteamericanos como Lonworks o CEBus.

Respecto al nivel físico el nuevo estándar, los protocolos KNX pueden funcionar sobre los medios que se detallan en la Figura 2-28:

- **TP0** (Par Trenzado, Tipo 0 – *Twisted Pair, Type 0*). Este medio, con una velocidad de 4800 bps, ha sido tomado de **BatiBUS**. Los productos certificados KNX TP0 diseñados para este medio pueden trabajar en la misma línea de bus que los componentes certificados por BatiBUS pero no son capaces de intercambiar información entre ellos.
- **TP1** (Par Trenzado, Tipo 1 – *Twisted Pair, Type 1*). Este medio, con una velocidad de 9600 bps, ha sido tomado del **EIB**. Los productos certificados EIB y KNX TP1 operan y comunican con cada uno de ellos en la misma línea de bus.
- **PL100** (Línea de Fuerza, 110 kHz - *Power-line, 110 kHz*). Este medio de corrientes portadoras sobre la red eléctrica, con una velocidad de 1200 bps, ha sido también asumido desde **EIB**. Los productos certificados EIB y KNX PL110 operan y comunican entre ellos en la misma red de distribución eléctrica, aprovechando la norma EIB equivalente.
- **PL-132** (Línea de Fuerza, 132 kHz - *Power-line, 132 kHz*). Con una velocidad de 2400bits/s, ha sido asumido desde **EHS**. Los componentes certificados KNX PL132 y EHS 1.3a, operan conjuntamente en la misma red de distribución eléctrica pero no se comunican entre ellos sin un convertidor de protocolo exclusivo.
- **RF** (Radio Frecuencia a 868 MHz – R*adio Frequency on 868 MHz*). Con una velocidad de 38,4 Kbps ha sido desarrollado directamente en la estructura del estándar **KNX**.
- **Ethernet** (KNX sobre IP – *KNX-over-IP*). Permite el envío de telegramas KNX encapsulados en telegramas IP. Se ha aprovechado la norma **EIB.**net.

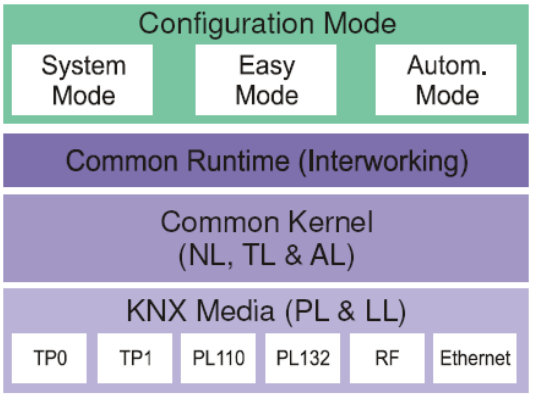


Figura 2-28. Medios físicos en el entorno KNX.

Por otra parte, KNX incorpora nuevas formas de configuración (véase la Figura 2-28) sobre el ahora llamado "modo de sistema" que era el empleado en EIB. A este modo de configuración hay que añadir el "modo fácil" y el "modo automático" con el que se facilita tanto a los profesionales como al usuario final, la puesta en marcha, adaptándose a los conocimientos técnicos de cada cual, sin restar por ello posibilidades al sistema. Fruto de estos planteamientos se puede afirmar rotundamente que KNX es compatible con los productos EIB previamente instalados, permitiendo una evolución de EIB a KNX totalmente operativa.

A día de hoy, más de 100 compañías de todo el mundo forman parte de Konnex. Una diferencia con sus predecesoras es que ya no es preciso ser fabricante de productos KNX para ser miembro de la misma sino que están incorporados en la asociación operadores de electricidad y telecomunicaciones, integradores, instaladores, etc. KNX es en estos momentos estándar mundial.

Ya venía avalado desde 2003 por parte de CENELEC (Comité Europeo de Estandarización Electrotécnica) con la aprobación de la norma EN-50090 y, desde 2007, la norma ISO/IEC 14543 da ámbito mundial a este estándar de automatización de viviendas y edificios.

En el mercado americano ha surgido una iniciativa similar, denominada SCP (*Simple Control Protocol*), pero se encuentra todavía en fase de definición. El *Simple Control Protocol* (SCP) es un intento del gigante Microsoft y de la mayor empresa del mundo (por facturación y empleados) General Electric, de crear un protocolo para redes de control que consiga afianzarse como la solución, de facto, en todas las aplicaciones de automatización de edificios y viviendas. Se trata de poner un poco de orden en la oferta que hay ahora mismo en EEUU para estos temas (X-10, CEBus, Lonworks, otros) y auspiciar la convergencia de todos estos hacia un protocolo abierto y libre de royalties, además de desarrollar un conjunto de productos que cubran todos los requisitos de automatización de las viviendas. Esta iniciativa, aunque ya había trabajos previos, tiene formalmente apenas dos años de vida.

2.6 Evolución de la Domótica en el Mercado Español

En la actualidad, en la oferta se aprecia una tendencia en el diseño de nuevos sistemas domóticos que se enfoca a la descentralización de funciones. Esto se debe a la demanda de los usuarios de sistemas cada vez más simples, que puedan gestionar todos los aspectos del hogar pero que a la vez permitan su escalabilidad a voluntad del usuario [Mercahome 04].

2.6.1 Evolución y Mercado Potencial

Los primeros estudios realizados en los inicios de este sector sugirieron expectativas muy importantes de crecimiento, dadas las ventajas aportadas por la Domótica: ahorro energético, confort, conectividad y seguridad. Sin embargo, no se han cumplido las expectativas iniciales por diversos motivos, entre otros por la situación por la que pasó el sector de la construcción a principios de los años noventa. A pesar de ello, esta disciplina ha mantenido una evolución ascendente prácticamente constante, con un notable incremento en los últimos años [Mercahome 04][Mint 08].

En la Figura 2-29 se presenta la evolución de la implantación y crecimiento de la domótica en paralelo al número de viviendas construidas hasta el año 2008, en el que se instalaron más de 50.000 sistemas domóticos en viviendas de nueva construcción [Mint 08]. Para los próximos seis años se estima que las nuevas promociones ascenderán a 5 millones de viviendas y se prevé que la penetración de la domótica alcanzará el **35% de las nuevas promociones**, equiparándose al porcentaje actual europeo, lo que dará una cifra aproximada de 1,7 millones de nuevas viviendas domóticas en 2014.

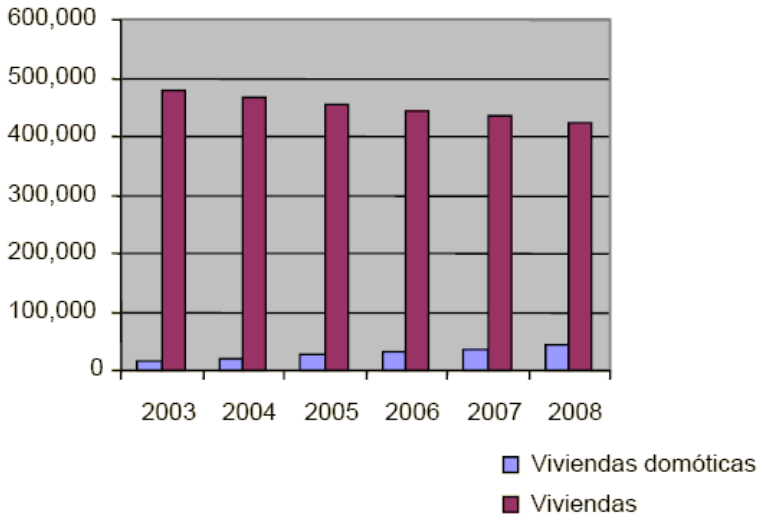


Figura 2-29. Evolución experimentada por la Domótica hasta 2008. Fuente: Telefónica I+D.

Algunos aspectos significativos en la evolución de la implantación de los sistemas domóticos han sido:

- La desaceleración del mercado de la construcción, que busca alternativas para diferenciar su producto. La crisis del sector inmobiliario, iniciada en el año 2008, propiciará un incremento del 30% en el sector domótico para el año 2010[3].
- La creación de nuevas empresas que operan de forma exclusiva en el sector.
- La regulación del mercado.
- Los costes de algunos productos de nuevo diseño se han ido reduciendo al combinar sinergias.
- Las acciones que las diferentes asociaciones de domótica (CEDOM, Institut Cerdà, y otras) han realizado en los últimos años con el objetivo de formar, informar e impulsar el desarrollo de este mercado en España.
- La aparición de diversos medios de comunicación con la misión de formar, informar y difundir los eventos y los nuevos avances en aspectos de domótica.
- La aparición de importantes iniciativas para la normalización de la domótica. A nivel europeo *Foro SmartHouse* y la aparición de la normativa EN50090. A nivel Nacional el AEN/CTN 205.
- Los sistemas domóticos representan un coste razonable respecto al coste de "producción" de la vivienda, entre un 5% y un 8% del coste de construcción bruto, que corresponde desde un 0,5 hasta un 2% del coste de compra de la vivienda, y además no suponen un problema técnico desde la perspectiva del Constructor.
- La estimación de costes es claramente factible mediante las especificaciones propias de un proyecto técnico.
- Se ha producido un aumento significativo en la **demanda por parte de los usuarios finales**.

2.6.2 El Mercado Actual

Si nos centramos en la demanda actual del mercado, ésta se puede abordar desde diferentes puntos de vista. En el ámbito del área de aplicación, la domótica ocupa un lugar relevante dentro de las principales áreas demandadas en las viviendas (domótica y seguridad), encontrándose por detrás de los sistemas de seguridad en el porcentaje total de sistemas instalados (8% frente a un 14%), pero muy por delante en lo que al valor total de los sistemas domóticos se refiere (casi cuatro veces más), tal como se puede observar en la Figura 2-30.

[3] http://www.prensa.ifema.es/SalaPrensa/img_noticia/14Matelec08SectorDomoticaCedom(16.06.08).doc

Otro aspecto relevante es el tipo de viviendas en que se instalan los sistemas domóticos. Aproximadamente un 85% de las instalaciones domóticas se realizan en viviendas de nueva construcción (véase la Figura 2-31). Este porcentaje es tan elevado debido, principalmente, a la apuesta que están realizando las promotoras, que ven la domótica como una solución a las necesidades de los usuarios a los que van destinadas las viviendas, pero sobre todo, como una manera de diferenciar su producto de la competencia.

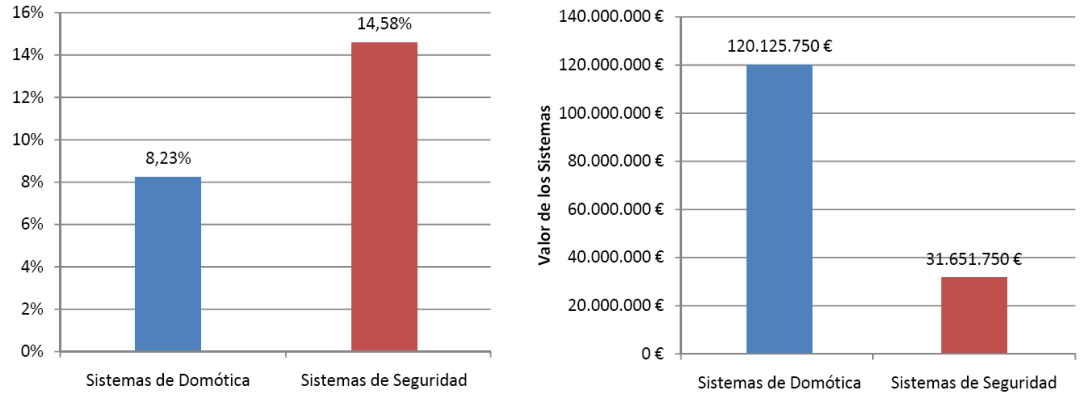


Figura 2-30. Distribución de la demanda por áreas en 2007 [Mint 08].

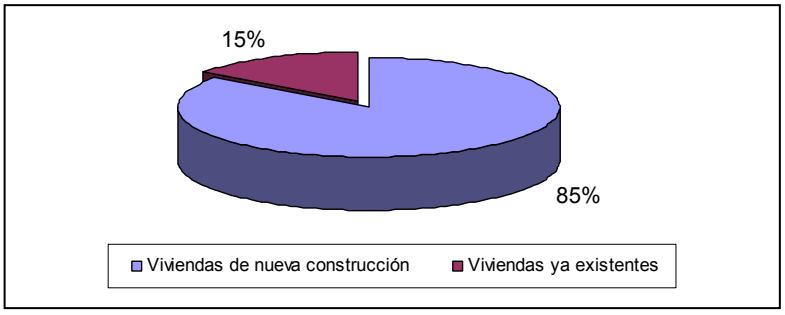


Figura 2-31. Demanda de domótica según el tipo de obra [Mercahome 04].

Por otra parte, el bajo porcentaje en rehabilitación viene dado por el desconocimiento de la mayoría de los usuarios en referencia a la domótica y por la creencia heredada de los primeros años de que todo lo referente a esta disciplina implica una complejidad de instalación y unos costes elevados, lo que dificulta su implantación en edificios ya construidos. Por ello, la implantación de la domótica en viviendas existentes no rehabilitadas es insignificante.

Otro dato importante es el conocimiento de la demanda según la arquitectura del sistema domótico. En la Figura 2-32 se puede apreciar que el tipo de sistema más demandado es el centralizado con un 42% (suele tratarse de sistemas propietarios de muy bajo coste y escasas prestaciones). También se puede observar cómo los sistemas distribuidos tienen un peso importante en la demanda (con un 32%), hecho que

concuerda con las tendencias actuales, y se prevé que su demanda sea cada vez mayor en los próximos años.

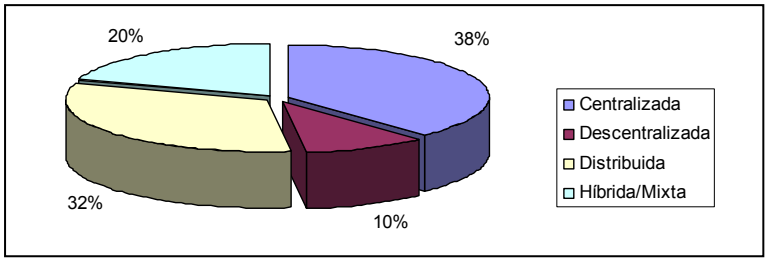


Figura 2-32. Porcentaje de demanda en 2007 según la arquitectura de los sistemas domóticos [Mint 08].

En lo referente a los protocolos utilizados, el estudio de [Mint 08] sitúa a KNX/EIB a la cabeza de los sistemas abiertos, con un 20% de demanda, por delante de otros protocolos como Lonworks (9%) y a la par de X10 (20%). La elevada demanda de X10 se debe a que se instala en gran parte de la obra rehabilitada, y cuando hablamos de obra nueva, quedan incluidos algunos sistemas globales que dentro de sus aplicaciones utilizan el X10 para dar solución a las necesidades exigidas por el cliente.

2.7 Conclusiones y Aportaciones a la Tesis

La domótica es una disciplina reciente que integra numerosos servicios y sistemas relacionados con la gestión de viviendas y edificios. En el desarrollo de los sistemas domóticos se ven involucradas diversas materias como la automatización, tecnologías de la información, gestión de redes o la programación de microprocesadores.

En los últimos años han surgido numerosos sistemas y estándares domóticos, y aunque siguen evolucionando, los recientes procesos de convergencia de los principales estándares del mercado demuestran que se está alcanzando cierta madurez, lo que permitirá dirigir un mayor esfuerzo a su mejora, más que a la búsqueda de nuevas soluciones tecnológicas o arquitectónicas. En este sentido, los sistemas mejor posicionados son Konnex (resultado del proceso de convergencia de los principales sistemas europeos) y Lonworks, una de las tecnologías con mayor implantación a nivel mundial. Este aspecto se tendrá en cuenta a la hora de seleccionar la(s) plataforma(s) sobre la(s) que realizar las aplicaciones.

Esta evolución en las tecnologías domóticas ha sido paralela a un crecimiento casi exponencial de la demanda de estos sistemas en el sector doméstico y terciario.

En este capítulo se han realizado varias aportaciones importantes para el planteamiento de esta Tesis Doctoral. En primer lugar, se ha concretado con precisión el dominio de aplicación de los sistemas domóticos, lo que permitirá abordar en capítulos posteriores la definición de los servicios necesarios en este tipo de aplicaciones.

En segundo lugar, se han revisado las características tecnológicas propias de los sistemas domóticos y se han estudiado para diferentes soluciones (la mayoría recogidas por estándares internacionales). La comprensión de esta tecnología y de las soluciones existentes es fundamental para entender el proceso de diseño que se sigue para desarrollar sistemas domóticos y los problemas que conlleva.

En la actualidad, el proceso de diseño en el campo de la domótica es similar al empleado en otros sistemas que interactúan con el entorno, como los de automatización y control industrial, la robótica o los sistemas de inspección automatizados. En todos ellos es necesaria la intervención de un especialista del dominio que tiene una amplia experiencia en la plataforma sobre la que se realizará la implementación. Además, en la mayoría de los casos se realiza el diseño prácticamente desde cero, y se requiere un gran esfuerzo para la generación del código en el lenguaje de programación que se vaya a utilizar, conduciendo a soluciones a medida que rara vez son reutilizadas. Estos y otros muchos problemas plantean la necesidad de una mejora en el enfoque utilizado en el proceso de desarrollo.

En el caso particular de los sistemas domóticos, el componente software es importante, puesto que han de programarse dispositivos o nodos (sensores, actuadores y controladores) en una arquitectura centralizada o distribuida. Por ello, la aplicación de técnicas de Ingeniería del *Software* puede contribuir a mejorar el proceso de desarrollo de estos sistemas.

En esta Tesis se va a utilizar un planteamiento que está revolucionando la Ingeniería del *Software* en los últimos años, el desarrollo dirigido por modelos (MDE: *Model Driven Engineering*), que se presentará en el siguiente capítulo. Su aplicación permitirá definir una metodología que abarque todo el ciclo de vida en el desarrollo de sistemas domóticos, mejorando el proceso en su conjunto. Entre los beneficios que se esperan obtener están:

- Aumento del nivel de abstracción, tratando el problema en términos del dominio y de manera totalmente independiente de la plataforma final de implementación.
- Mejora de la productividad, al realizar el proceso de generación de código de manera automática o semiautomática.
- Reutilización de las soluciones desarrolladas.
- Posibilidad de trasladar una misma solución a una nueva plataforma sin tener que realizar de nuevo la especificación.
- Reducción de errores en el proceso de desarrollo y reducción de la necesidad de personal especializado.
- Posibilidad de integrar otras herramientas para tareas de validación, animación, etc.

Finalmente, se ha justificado el interés de los sistemas domóticos en la sociedad actual, que está demandando la implantación de las nuevas tecnologías de la información en la vida cotidiana con el fin de aumentar el confort, optimizar el consumo, mejorar la seguridad e integrar todos los dispositivos con las redes de comunicación existentes y futuras.

CAPITULO 3

Desarrollo Dirigido por Modelos (MDE)

Este capítulo presenta el novedoso enfoque MDE (Model Driven Engineering) de desarrollo de software dirigido por modelos, que utiliza los modelos como artefacto principal en todo el proceso de diseño software. MDE proporciona una nueva teoría de desarrollo y una serie de herramientas que soportan su aplicación con el fin de aumentar el nivel de abstracción para la realización del diseño del software y obtener de forma automática o semiautomática las diferentes representaciones del mismo y el código final ejecutable.

El capítulo se organiza en siete secciones. La primera introduce la evolución de la Ingeniería del Software desde la tecnología procedural, la orientación a objetos y el porqué es necesario un nuevo enfoque en su desarrollo. La siguiente presenta el modelado de sistemas como actividad humana esencial y los conceptos más relevantes en el campo de los sistemas y modelos. La tercera sección describe las ideas básicas del desarrollo de software dirigido por modelos. Las dos secciones siguientes presentan la propuesta de la OMG para MDE, denominada MDA, y las tecnologías que dan soporte a MDE. Finalmente, la última sección revisa el estado del arte de la utilización de la metodología MDE en el desarrollo de sistemas reactivos, prestando especial atención a su aplicación a sistemas domóticos.

3.1 Motivación

La evolución de la Ingeniería del *Software* se ha visto marcada por varios hitos a lo largo de su corta historia. Uno de los más importantes fue el giro, a principios de los años 80, que supuso el paso de la programación estructurada (lo importante eran las funciones) a la orientación a objetos (lo importante son los objetos como unión encapsulada de datos más funciones).

Pero en sus más de veinticinco años de historia la orientación a objetos ha alcanzado su madurez, dado que los requisitos impuestos por los nuevos sistemas están superando las posibilidades de este enfoque. Las plataformas evolucionan a gran velocidad, incrementándose el volumen de datos y de código, así como los aspectos funcionales y no funcionales de las aplicaciones. Por otra parte, es cada vez mayor la heterogeneidad en elementos clave como los lenguajes y paradigmas utilizados, los protocolos de acceso y de manipulación de datos, los sistemas operativos y las plataformas *middleware* empleadas. Además, aparecen nuevas tecnologías a una velocidad creciente y las previsiones indican que este crecimiento se va a mantener o incluso incrementar. Y para empeorar aún más las cosas, las tecnologías existentes no desaparecen, sino que se ocultan en capas de software más profundas.

Todos estos factores han conducido a la conclusión de que **la orientación a objetos es insuficiente** [Astrachan 05], ya que presenta deficiencias importantes en los siguientes aspectos:

- El enfoque orientado a objetos no afronta adecuadamente los requerimientos de computación presentes y futuros.
- Los lenguajes orientados a objetos han perdido la simplicidad (o "pureza") que los hacía especiales, y que era la fuente de su expresividad y potencial de desarrollo.
- Conceptos tan importantes como la encapsulación, que se introdujeron para reducir la influencia de los programadores en el desarrollo de software, fallan cuando se necesita describir propiedades globales o se requiere que el *software* evolucione o se someta a cambios importantes.
- Una de las expectativas que llevó al planteamiento de la orientación a objetos fue la reutilización, pero la experiencia ha demostrado un éxito más que moderado en este aspecto.

De la promesa inicial de simplicidad (véase la Figura 3-1) se ha pasado a una complejidad creciente. La tecnología de objetos planteaba en sus inicios tres conceptos básicos: objetos, clases y métodos. Pero en su evolución a la tecnología de componentes, los conceptos manejados para satisfacer las necesidades de las aplicaciones software crecen de manera exponencial. Además, no está clara la correspondencia entre los conceptos de la orientación a objetos desde la definición de los requisitos hasta la implementación.

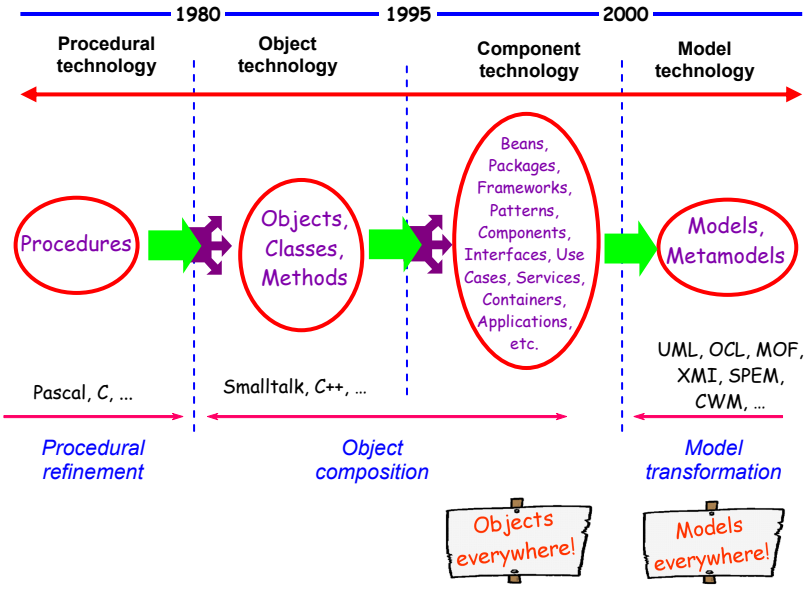


Figura 3-1. Evolución de la Ingeniería del *Software*, adaptada de [Bézivin 05c].

Por otra parte, quedan aún por resolver muchos conceptos que no terminan de encajar en la orientación a objetos. Por ejemplo, los servicios, desde los casos de uso más elementales hasta los servicios *web* avanzados, no pueden ser fácilmente representados mediante objetos, de hecho, un servicio no es un objeto. Lo mismo ocurre con los aspectos, que describen propiedades transversales de la funcionalidad básica de un sistema, y que no pueden ser capturados de manera natural mediante programación procedural ni orientada a objetos. Otros ejemplos son los *plugins* y métodos, que tampoco son objetos, y la dificultad para representar bases de datos, procesos, transacciones, etc.

Por otro lado, el aumento de la complejidad y la aparición de nuevas tecnologías ha conducido a una guerra de *middlewares* orientados a objetos y a componentes (COM, DCOM, http, HTML, XML, CORBA, FJB, Java de Sun, C# y .NET de Microsoft, …) en la que no existe un claro ganador. Parece que la próxima batalla se librará en el campo de la transformación de modelos, donde la iniciativa MDA (*Model Driven Architecture*) del OMG (*Object Management Group*), que se fundamenta en el uso de modelos, se perfila como una solución integradora para el desarrollo de *software* [Schmidt 06].

La tecnología de objetos ha satisfecho algunas de las expectativas que motivaron su adopción, pero ha fallado en la consecución de muchas otras. Quizás una de las causas se puede encontrar en que se ha dejado de buscar la generalidad mediante la unificación.

A principios de esta década emerge un nuevo paradigma: la ingeniería dirigida por modelos (MDE: *Model Driven Engineering*) [Schmidt 06][Bézivin 05b][Sendall 03] con el propósito de suponer el paso definitivo hacia la industrialización del software, o al menos proporcionar mejoras significativas en la productividad y calidad. Éste será un paso revolucionario en el desarrollo del software, comparable al salto cualitativo

[Mellor 04] que supuso el diseño de los primeros compiladores de FORTRAN en 1957. MDE se encuentra en la actualidad en la etapa inicial de generación de expectativas que prometen solucionar todas las deficiencias detectadas en la madurez de la orientación a objetos.

El enfoque MDE se basa en la utilización de modelos como elemento principal en todo el ciclo de desarrollo del software. El uso de modelos aumenta el nivel de abstracción, ya que permite centrarse en los conceptos, dejando en un segundo plano los detalles de implementación. Mediante la utilización de un conjunto de herramientas, los modelos se transforman a otros modelos y finalmente a código ejecutable, que se genera de manera automática o semiautomática.

Bézivin [Bézivin 05a] establece el paralelismo entre las premisas básicas de la orientación a objetos y MDE. Tal como se muestra en la Figura 3-2, mientras que la orientación a objetos ("todo es un objeto") se basa en relaciones de *instanciación y herencia*, MDE descansa sobre el principio "todo es un modelo", donde las relaciones básicas son de *representación* (un modelo *representa* un sistema) y de *conformidad* (un modelo es *conforme* a su meta-modelo).

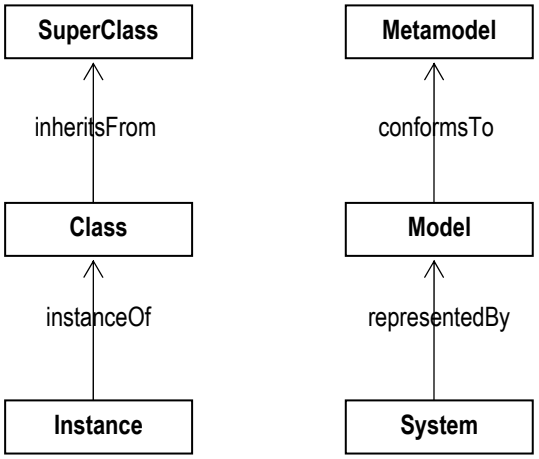


Figura 3-2. Nociones básicas en las tecnologías de objetos y modelos.

Al igual que las relaciones de *herencia* e *instancia* lo fueron en su día para la orientación a objetos, las relaciones de *conformidad* y *representación* son fundamentales para comprender el enfoque MDE. En las secciones posteriores se profundizará más en todos estos conceptos. Se está produciendo, por tanto, una transición de objetos a modelos, y sólo la evolución de esta nueva tecnología en los próximos años permitirá evaluar si realmente se cumplen todas las expectativas que está generando.

En la sección siguiente se hace un desglose exhaustivo de los principales conceptos usados en la orientación a modelos.

3.2 Modelado de Sistemas

El modelado es esencial a la actividad humana ya que casi toda acción viene precedida de la construcción, implícita o explícita, de un modelo. Los modelos se han utilizado desde la antigüedad con el fin de mejorar el entendimiento de los problemas y sus posibles soluciones. Por ejemplo, la técnica médica de la sangría, estaba basada en el modelo (incorrecto) que Hipócrates (año 460 a. C. - 370 a. C.) creó para el cuerpo humano, basado en el equilibrio de los cuatro humores: sangre, flema, bilis y bilis negra. Así, las enfermedades se suponían efecto de una alteración en este equilibrio, que se restauraba mediante sangrías. Si el modelo es incorrecto, la acción puede ser inapropiada (George Washington murió tras ser tratado de laringitis mediante una sangría prolongada) [Rothenberg 89]. La importancia del concepto de modelo nos lleva a analizar con detalle las distintas perspectivas que se pueden tener de él.

3.2.1 Definición de Modelo

Entre las definiciones de *modelo* existentes en la bibliografía reciente, destacan las siguientes:

- Un modelo es un conjunto de declaraciones sobre un sistema bajo estudio [Seidewitz 03].
- Un modelo es una descripción de (una parte de) un sistema y está escrito en un lenguaje bien definido. Un lenguaje bien definido tiene una forma (sintaxis) y un significado (semántica) correctamente especificados, de manera que puede ser interpretado automáticamente por una máquina [Kleppe 03].
- Un modelo es una simplificación de un sistema que ha sido construida con un objetivo en mente. Un modelo debe comportarse igual que se comportaría el sistema que modela [UML v2.1.1 07].

Pero quizás una de las definiciones más completas es la propuesta por [Rothenberg 89]:

- Modelar, en el sentido más amplio de la palabra, es el uso, con el fin de economizar, de algo en lugar del original para algún propósito cognitivo. Nos permite utilizar algo más simple, seguro y barato que la realidad para un propósito dado. Un modelo representa la realidad para dicho propósito; el modelo es una abstracción de la realidad, ya que no representa todos sus aspectos. Esto permite tratar con el mundo de una forma simplificada, evitando la complejidad, peligro e irreversibilidad de la realidad.

Esta definición pone de manifiesto tres atributos importantes del modelado: la referencia, el propósito y la reducción de costes:

- Un modelo se **refiere** siempre a un sistema (su referente). El referente debe estar bien definido y ser verificable de manera objetiva (aunque no tiene por qué existir, se puede modelar realidad virtual).

- El modelo se diseña con un **propósito** concreto, que puede ser la comprensión o manipulación de su referente, predicción, adquisición de experiencia, etc. El propósito determina qué aspectos del modelo se van a modelar y con qué fidelidad.
- Finalmente, la **economía** o reducción de costes es fundamental. Es más económico utilizar el modelo para el propósito que se ha diseñado que usar el sistema que modela (por ejemplo, es mucho más económico y viable estudiar el reparto de cargas de un puente con un modelo, que construyendo el puente real).

Bran Selic presenta una visión similar [Selic 03], pero introduce algunos matices interesantes afirmando que para que un modelo sea realmente útil y efectivo, debe cumplir cinco propiedades básicas:

- **Abstracción**. El modelo ofrece una **vista parcial** y simplificada de la realidad, de modo que nos ayuda a comprenderlo mejor eliminando los detalles de menor importancia. Pero no conviene reducirlo demasiado, ya que el modelo obtenido puede llegar a no representar la realidad. Por ello, en muchos casos, para modelar un sistema son necesarios varios modelos desde diferentes puntos de vista.
- **Inteligibilidad**. Además de representar correctamente la realidad, el modelo debe ser inteligible por los usuarios. Un modelo difícil de comprender puede denotar una falta de precisión o abstracción del mismo.
- **Precisión y previsibilidad**. La representación que se hace de la realidad debe ser lo suficientemente precisa como para poder predecir algunas propiedades no obvias ni modeladas directamente por él mediante la ejecución del modelo o análisis formales.
- **Economía**. El modelo debe ser mucho más económico en términos de construcción y análisis que el sistema modelado.

3.2.2 Modelos y Sistemas

Existe una gran variedad de modelos de sistemas que podemos encontrar en muy diferentes ramas de la ciencia. Algunos ejemplos son los modelos matemáticos, hidrológicos, biológicos, ecológicos, económicos, meteorológicos, de simulación, predictivos, etc.

Cuando se habla de modelos y sistemas conviene hacer algunas puntualizaciones. En primer lugar, un modelo es una representación de (*repOf*) un sistema concreto (o siguiendo los postulados de Bézivin, recogidos en la Figura 3-1, el sistema está representado por (*representedBy*) un modelo). Por lo tanto, el modelo **no es el sistema** real, aunque muchas veces se puede llegar a confundir.

Una de las características básicas de los modelos es que son una abstracción **o vista parcial** del sistema desde un punto de vista concreto, por lo que para modelar un

sistema en muchos casos son necesarios **varios modelos (vistas parciales)**. Un ejemplo ilustrativo es el del cuerpo humano, representado por modelos del esqueleto, respiratorio, muscular, nervioso, circulatorio, digestivo y endocrino, entre otros muchos (véase la Figura 3-3). La idea de abstracción o vista parcial es fundamental, puesto que facilita su comprensión (es posible comprender los sistemas muscular o esquelético por separado, pero la vista conjunta de todos los sistemas resultaría casi imposible de comprender para la mente humana). Además, cada vista parcial del sistema tendrá un vocabulario y un conjunto de técnicas y herramientas específicas del mismo.

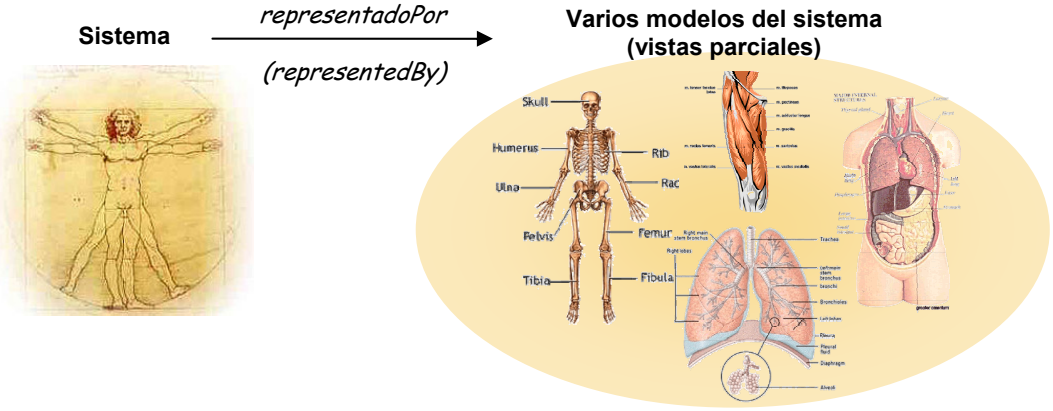


Figura 3-3. Diferentes modelos o vistas del cuerpo humano.

Otro ejemplo que ha dado lugar a una amplia reflexión en la práctica del modelado es el de los mapas geográficos (véase la Figura 3-4). Su estudio ha llevado a la conclusión de que la precisión del modelo es algo relativo, ya que la finalidad del modelo no es captar todos los aspectos del sistema, sino sólo aquellos de mayor relevancia para su comprensión, por lo que suelen ser necesarias varias vistas, cada una de las cuales capta aspectos específicos.

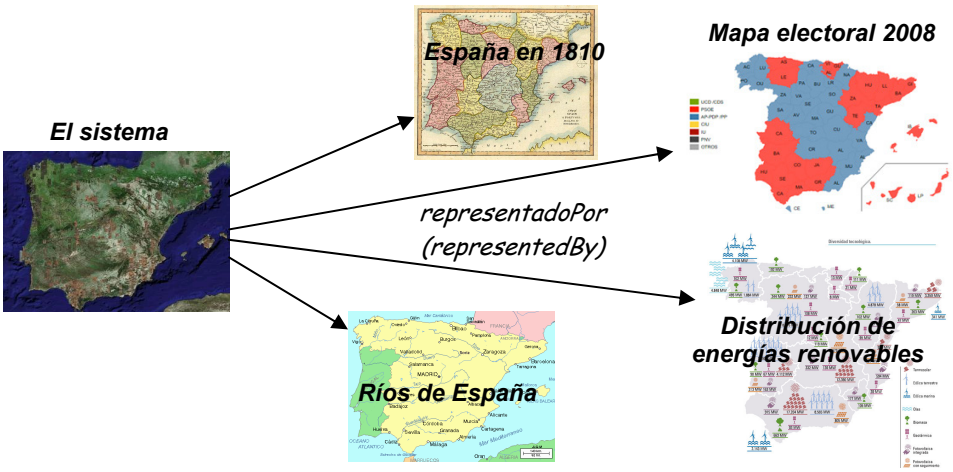


Figura 3-4. Modelado mediante mapas.

En el caso del software, éste es un modelo complejo y compuesto que puede representar diferentes aspectos de un sistema con distintos niveles de abstracción. En función del nivel de abstracción, el software puede ser una representación de los requisitos de la aplicación, la plataforma de ejecución o la lógica de negocio. Además, está escrito en un lenguaje de programación (conforme a un lenguaje).

Otro aspecto importante en la construcción de modelos de sistemas es la **finalidad** con la que éstos se diseñan. Con frecuencia interesa realizar un modelo para, a partir de éste, construir el sistema. Por ejemplo, antes de construir una casa, se diseñan los planos (modelos) de distribución, cimentado, fontanería, electricidad, etc. Estas diferentes vistas, cada una de ellas expresada con una terminología propia (o lenguaje específico de cada dominio), recogerán todas las características relevantes para obtener el sistema final. En otras ocasiones los modelos se construyen a partir de sistemas ya existentes para comprenderlos mejor (modelos geográficos, del cuerpo humano, de organizaciones, etc.). Aún hoy existen numerosos modelos cuya construcción es un reto de gran interés, como el modelo del genoma humano o de las funciones cerebrales.

La **naturaleza** del sistema es también un factor que condiciona las técnicas de extracción del modelo y las características de éste. La Figura 3-5 ilustra un ejemplo de cómo diferentes tipos de sistemas permiten la extracción de distintos tipos de modelos; si el sistema es estático, el modelo extraído puede ser una película, pero si es estático su representación podría ser un dibujo. Si además de estático, es tridimensional, se podrá extraer un modelo mediante una fotografía, pero si es bidimensional, se podrá hacer mediante una fotocopia. Si contiene texto, se podrá realizar una interpretación mediante reconocimiento de caracteres, y si ese texto es un código fuente en Java, se podría hacer ingeniería inversa para obtener el modelo.

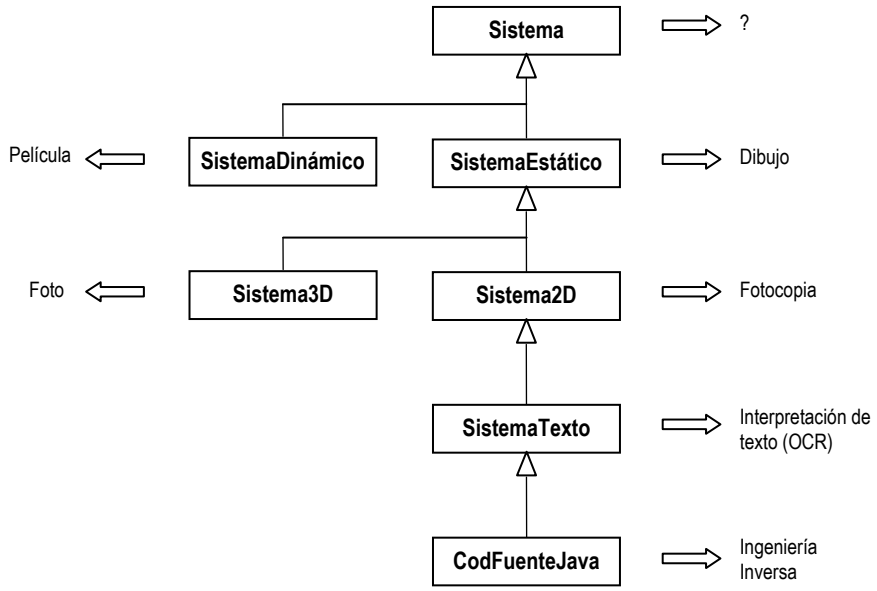


Figura 3-5. Técnicas de extracción dependiendo de la naturaleza del sistema. Idea extraída de [Bézivin 05c].

Una de las características más importantes de los sistemas y modelos, es su condición temporal. Cuando los sistemas son estáticos (no cambian en el tiempo), sus modelos son obviamente estáticos (por ejemplo, los mapas). Por el contrario, muchos sistemas son dinámicos, pero en numerosas ocasiones sus modelos son estáticos. Esta es una característica muy interesante de los modelos, porque facilita su comprensión; un diagrama de estados es un modelo estático de un sistema dinámico. Sin embargo, en otras ocasiones los modelos evolucionan en el tiempo (por ejemplo un simulador, o el código fuente automodificable).

En la Figura 3-6 se resumen los tipos de sistemas y modelos por su condición temporal. La única relación que no tiene sentido es la de un modelo dinámico de un sistema estático, siendo la relación de modelos estático de sistema dinámico. Independientemente del tipo de sistema o modelo (estático o dinámico) podemos concluir que la relación de representación equivale a la *sustituibilidad contextual*, es decir, el modelo debería ser capaz de responder un conjunto de cuestiones de la misma manera que lo haría el sistema real.

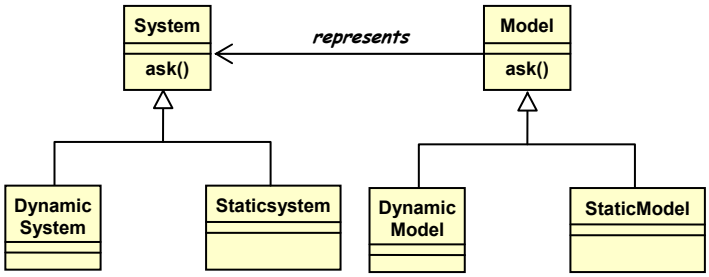


Figura 3-6. Relaciones entre modelos y sistemas estáticos y dinámicos [Bézivin 05a].

3.2.3 El concepto de Metamodelo

En los ejemplos descritos en el apartado anterior se han revisado las principales características relativas a la actividad del modelado y la relación de **representación**, recogida en las nociones básicas de Bézivin (véase la Figura 3-2). Siguiendo la analogía objetos-modelos, en el modelado existe una segunda relación, la **conformidad**, que constituye uno de los pilares básicos de esta tecnología.

Si recuperamos el ejemplo de modelado mediante mapas recogido en la Figura 3-4, podemos ver cómo éstos representan de manera gráfica el sistema, pero además incluyen una información adicional que es fundamental para su comprensión: la *leyenda*. Un mapa sería inútil sin su leyenda, puesto que es la que nos dice **cómo interpretarlo:** si perdemos la leyenda el mapa sería imposible de interpretar. También existe información implícita, como la dirección (la parte superior representa el norte), la representación del mar en color azul, etc.

Decimos que un mapa es **conforme** a su leyenda (véase la Figura 3-7), es decir, que el mapa está escrito en un lenguaje gráfico definido por su leyenda. Esta relación se

puede generalizar a todos los modelos: un modelo es siempre conforme a su metamodelo (principio recogido en la Figura 3-2).

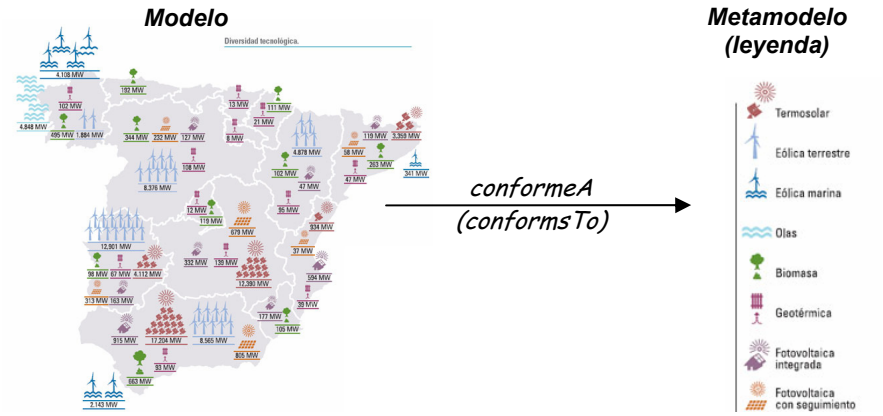


Figura 3-7. Relación de conformidad entre modelo (mapa) y metamodelo (leyenda).

El concepto de metamodelo es uno de los pilares sobre los que descansa la metodología dirigida por modelos, y por tanto, uno de los fundamentos básicos para la actividad del modelado. De entre las numerosas definiciones existentes en la literatura, se pueden destacar:

- Un metamodelo es un modelo que define el lenguaje para definir un modelo [MOF 04].
- Un metamodelo es un modelo de especificación para una clase de sistemas que cumplen que cada uno de ellos es en sí mismo un modelo válido en un cierto lenguaje de modelado [Seidewitz 03].
- Un metamodelo es un modelo que define la estructura, semántica y restricciones de una familia de modelos [Mellor 04].

De estas definiciones se deduce que un metamodelo recoge todos los conceptos de un dominio de sistemas y las relaciones entre ellos que son importantes para el usuario. De este modo se garantiza la conformidad entre un modelo y su metamodelo.

A su vez, el metamodelo no tiene por qué ser único. Para un dominio pueden existir varios metamodelos que permitan describirlo desde varios puntos de vista y con diferentes niveles de detalle. Los modelos de los mapas o las vistas del cuerpo humano (véase la Figura 3-3 y la Figura 3-4) estarán expresados conforme a diferentes metamodelos.

Finalmente, conviene puntualizar, revisando las relaciones básicas descritas en la Figura 3-2 para las tecnologías de objetos y modelos, que el modelo de un modelo no es un metamodelo (una discusión parecida se generó con la relación de instancia y herencia en orientación a objetos). Un modelo de un modelo sigue siendo un modelo, representación del anterior. La creación de modelos a partir de modelos es muy habitual y no debe confundirse con la relación de conformidad (modelo conforme a

metamodelo). Para aclarar esta idea, en la Figura 3-8 se muestra el famoso cuadro de Magritte en el que se lee *"Ceci n'est pas une pipe"* (esto no es una pipa), insistiendo en el hecho de que el cuadro que representa una pipa puede ser útil para muchos propósitos, pero no para fumar tabaco. Además, podemos tener un cuadro del cuadro, y así sucesivamente. La interpretación de "modelo de un modelo" está recogida por la relación de representación, mientras que un modelo está relacionado con su metamodelo por la relación de conformidad.

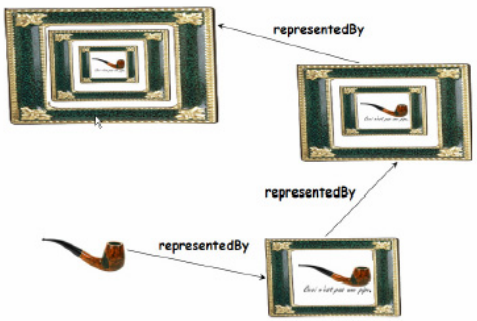


Figura 3-8. Cuadro de Magritte "Ceci n'est pas une pipe" [Bézivin 05a].

La siguiente sección profundiza en el enfoque MDE, que utiliza los modelos y metamodelos de forma intensiva a lo largo del ciclo de vida del desarrollo software.

3.3 Desarrollo de Software Dirigido por Modelos (MDE)

3.3.1 Introducción

El término "desarrollo o ingeniería dirigida por modelos" (MDE) hace referencia a la técnica que hace uso, de forma sistemática y reiterada, de modelos como elementos básicos a lo largo de todo el proceso de desarrollo [Kent 02]. MDE utiliza modelos como elementos de entrada y salida en todo el proceso. El empleo de modelos, que son una representación simplificada de la realidad, permite aumentar el nivel de abstracción en la realización del diseño y la reutilización de los mismos. Así, los artefactos generados son independientes del lenguaje y paradigma de programación que se utilizará en las etapas finales para la generación de código. Además, las ideas se expresan de forma explícita en términos propios del dominio del problema, y no diluidas en el código del programa.

Según [Bézivin 05a], MDE persigue, entre otros, la separación del modelo de negocio de la plataforma de implementación, la identificación, expresión precisa, separación y combinación de aspectos específicos del sistema en desarrollo con lenguajes específicos de dominio, el establecimiento de relaciones precisas entre los diferentes lenguajes en una arquitectura global, y en particular, la posibilidad de expresar transformaciones entre ellos.

3.3.2 Beneficios Esperados

Los beneficios que se esperan obtener mediante esta nueva metodología de diseño basada en modelos son [Selic 03]:

- Mayor velocidad en la implementación.
- Mejorar la calidad del código.
- Facilitar el mantenimiento.
- Desarrollo ágil (*Agile Development*).
- Incrementar la reusabilidad.
- Mejorar la flexibilidad y extensibilidad.

Una de las premisas básicas de MDE es la obtención (semi)automática de código a través de las transformaciones de modelos. Así, la cantidad de código que debe escribirse manualmente para una aplicación se reduce drásticamente, y el desarrollador debe centrarse únicamente en el código personalizado, que no suele superar el 10% del total (en las aplicaciones, en torno al 50% del código es genérico y el 40% semi-genérico). Esto repercute directamente en un aumento de la productividad.

La fiabilidad del código es otro de los atributos que mejora con el empleo de MDE. El código generado automáticamente requiere menos depuración y verificación, y además, los modelos no se utilizan sólo para la generación de código, sino que pueden producir como salida herramientas de comprobación utilizando restricciones escritas en lenguajes de definición de restricciones (OCL: *Object Constraint Language*). También se produce una mejora en la consistencia y la calidad, ya que las transformaciones de modelos a código se basan en patrones de diseño que permiten generar un código estructurado conforme a una arquitectura.

En el software el mantenimiento constituye una parte muy importante del coste total del desarrollo. Las aplicaciones deben ser capaces de responder y adecuarse a los cambios del modelo de negocio y de la tecnología. Cambios como añadir atributos a una clase personalizada, integrar recursos externos, reprogramar modificaciones en los casos de uso, etc., suponen una gran cantidad de trabajo que se puede reducir considerablemente mediante la generación de código dirigida por modelos, ya que se reduce la cantidad de código que ha de escribirse de forma manual.

Las características intrínsecas a MDE permiten además soportar el Desarrollo Ágil (*Agile Development*, recogido en el manifiesto ágil [Agile 01]), que se centra en dar una respuesta flexible a cambios en los requisitos, obtener software ejecutable y utilizable desde las primeras etapas y en conseguir una comunicación más estrecha entre los desarrolladores de software y el cliente. Así, con MDE se puede obtener software bien estructurado y operativo que se construirá de forma iterativa durante todo el desarrollo, y se agiliza la propagación de cambios en el código.

Por otra parte, la reusabilidad es una de las promesas de esta metodología. Los patrones de diseño pueden evolucionar en los niveles del modelo para definir cómo estructurar las clases del dominio y los componentes para las tareas estándar. Los modelos del dominio se pueden almacenar en librerías para ser reutilizados en nuevas aplicaciones. Además, el modelo de negocio se mantiene separado en todo momento de la arquitectura de la tecnología utilizada en la implementación.

A continuación se aporta una definición más completa de la idea de modelo, metamodelo y la relación existente entre ambos.

3.3.3 Definición Revisada de Modelo y Conformidad. Meta-metamodelos

Los modelos se pueden definir de forma estructural como multigrafos dirigidos. Un multigrafo dirigido $G = (N_G, E_G, \Gamma_G)$ consiste en un conjunto finito de nodos N_G, un conjunto finito de aristas E_G y una función de asignación de aristas a pares de nodos Γ_G: $E_G \rightarrow N_G \times N_G$. Así, un modelo $M = (G, \omega, \mu)$ es un triplete donde [Bézivin 05a] :

- $G = (N_G, E_G, \Gamma_G)$ es un multigrado dirigido.
- ω es un modelo, llamado modelo de referencia (metamodelo) de M, asociado al grafo $G_\omega = (N_\omega, E_\omega, \Gamma_\omega)$.
- $\mu: N_G \cup E_G \rightarrow N_\omega$ es una función que asocia elementos (nodos y aristas) de G a nodos de G_ω (metaelementos).

En el ejemplo de la Figura 3-9 (a) se muestran estas relaciones. *Client* es un modelo conforme al metamodelo donde se define el concepto de clase y atributo (*Client* tiene el atributo nombre de tipo *String*, aunque según el metamodelo podría tener muchos más). A su vez, el metamodelo es un multigrafo dirigido conforme a su meta-meta-modelo, que especifica un lenguaje gráfico para la descripción de metamodelos, según el cual, los metamodelos se componen de clases y asociaciones.

Según la relación de conformidad, todo modelo tiene un modelo de referencia (metamodelo) conforme al cual se ha especificado, pero el metamodelo también debe tener su propio meta-meta-modelo (que contiene los elementos básicos para describir metamodelos) y al que tiene que ser conforme. Además, el concepto de metamodelo no es absoluto, sino que depende del nivel de abstracción utilizado. Así, lo que en un determinado nivel es un metamodelo se convierte en un modelo para el nivel superior.

En la Figura 3-9 (b) y (c) se muestran estas relaciones: un modelo es siempre conforme a su modelo de referencia, pero además, ese modelo de referencia es a su vez un tipo de modelo. Según esto, se pueden distinguir tres tipos de modelos: terminales (su referencia es un metamodelo), metamodelos (su referencia es un metametamodelo) y metametamodelos (se suelen referenciar a sí mismos, es decir, son conforme a sí mismos). Todas estas relaciones se encuentran presentes en el ejemplo de la Figura 3-9 (a).

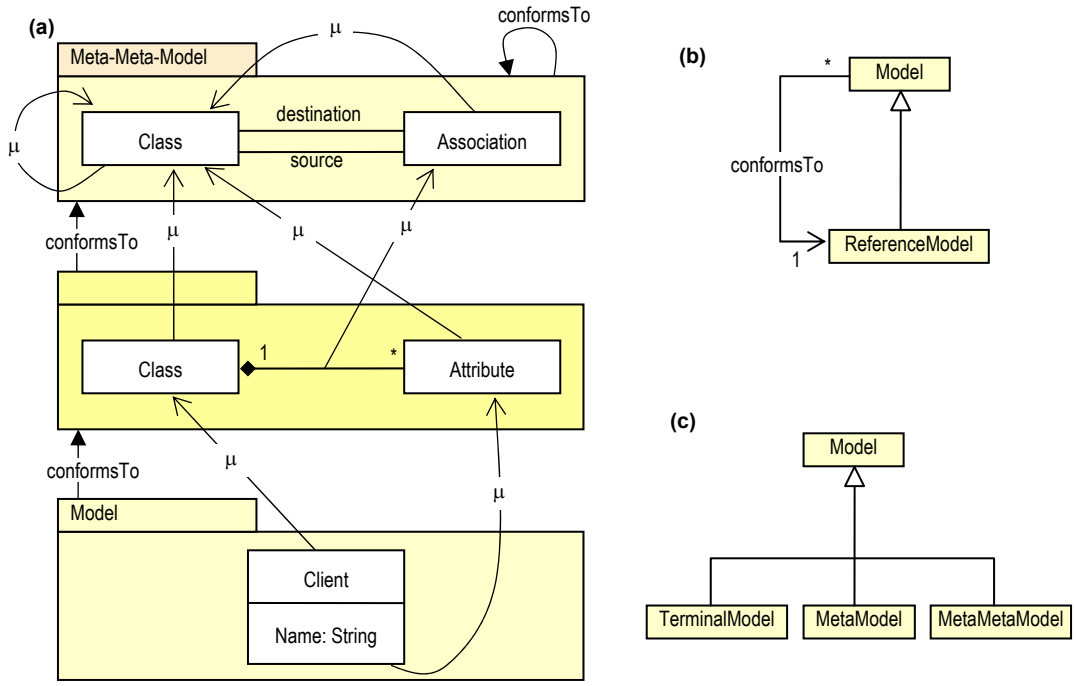


Figura 3-9. Relación entre modelos. (a) Multigrafos dirigidos para modelo y metamodelo y asociaciones de elementos (μ). (b) Relación de conformidad. (c) Tipos de modelos. (Extraído de [Bézivin 05a]).

Siguiendo el razonamiento de conformidad, se extrae una de las principales características de MDE: la relación de *recurrencia* en la definición de metamodelos. Para hacer posible la aplicación de MDE en la práctica, hay que romper en algún punto esta recurrrencia, lo que en MDE se consigue mediante la definición de un modelo recursivo. En un *modelo reflexivo* [Seidewitz 03] tanto el lenguaje de modelado como el metamodelo de dicho lenguaje coinciden y son el mismo, de modo se acaba con la recursividad de la definición. Esta es la solución adoptada por MDE (y en la versión MDA de la OMG utilizando MOF (*Meta Object Facility*, véase el apartado 3.4)).

3.3.4 Transformaciones de Modelos

En MDE las transformaciones de modelos son esenciales para dar soporte a todo el proceso de desarrollo. Las transformaciones permiten realizar conversiones de un modelo origen a otro destino entre diferentes metamodelos, manteniéndolos sincronizados. En el ámbito del desarrollo software esto permite, en una última instancia, convertir los modelos que utilizan los desarrolladores en modelos de la plataforma de implementación (código ejecutable).

En la bibliografía se pueden encontrar numerosas definiciones de transformación, entre ellas, las más clarificadoras son:

- Una transformación es la aplicación de una función de transformación para convertir un modelo en otro. Una función de transformación es un conjunto de reglas que definen cada uno de los aspectos de funcionamiento de una función de transformación [Mellor 04].

- Una transformación es la generación automática de un modelo a partir de otro modelo fuente, de acuerdo con una serie de reglas. Una definición de transformación es un conjunto de reglas de transformación que juntas describen cómo se transforma un modelo descrito en el lenguaje origen a un modelo descrito en el lenguaje destino. Una regla de transformación es una descripción de cómo se transforman una o más construcciones del lenguaje origen en una o más construcciones en el lenguaje destino [Kleppe 03].
- Una transformación genera un modelo destino a partir de un modelo fuente. Una vista es un tipo de transformación restringida que impone la restricción de que el modelo obtenido no puede ser modificado independientemente del modelo fuente [MOF-QVT 05].

Siguiendo la búsqueda de unificación en MDE (*todo es un modelo*), las transformaciones también se consideran modelos [Bézivin 01]. Por lo tanto, el modelo de la transformación deberá también ser conforme a un metamodelo. En la Figura 3-10 se esquematiza la transformación de modelos y las relaciones a diferentes niveles de abstracción. A partir de aquí se pueden extraer varias conclusiones:

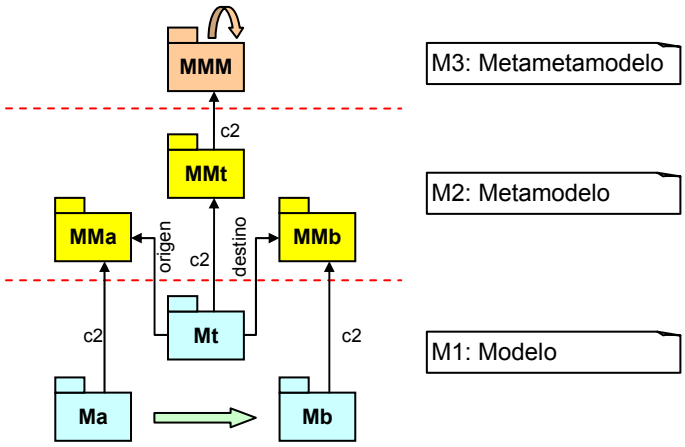


Figura 3-10. Transformación de modelos (extraído de [Bézivin 05a]).

- Una transformación genera un modelo destino Mb a partir de un modelo origen Ma. La transformación es en sí un modelo Mt.
- Puesto que los modelos pueden tener una representación visual, la consideración de las transformaciones como modelos proporciona la posibilidad de expresarlas de forma gráfica.
- Como todo modelo es conforme a un metamodelo, los modelos fuente y destino serán conformes a los metamodelos MMa y MMb. De forma análoga, la transformación Mt: Ma → Mb es también conforme a un metamodelo MMt que define el lenguaje común de transformación de modelos.
- Esto abre la posibilidad de realizar transformaciones de más alto nivel, es decir, transformaciones que tomen otras transformaciones como entrada y generen transformaciones como salidas.

- Las transformaciones no tienen por qué ser uno a uno, ya que es posible generar varios modelos a partir de un modelo único, o bien unificar modelos, es decir, generar un único modelo a partir de varios modelos de entrada (por ejemplo fusión de varias vistas de un mapa en uno sólo).

Existen diversas tecnologías para realizar la transformación de modelos: manipular directamente el modelo, utilizar una representación intermedia o emplear lenguajes de transformación. Inicialmente Lemesle [Lemesle 98] propuso extender el metametamodelo (MOF) con recursos básicos para transformaciones, aunque según Sendall [Sendall 03], el empleo de lenguajes específicos es lo más adecuado, siendo deseable que estos lenguajes ofrezcan abstracciones intuitivas y que abarquen el mayor número posible de situaciones.

Las características más importantes que deben tener las transformaciones de modelos son, según Mens [Mens 06], las siguientes:

- **Automatización**. Es necesario definir mecanismos para programar transformaciones automáticas a partir de una serie de modelos origen, pero también debe contemplarse la posibilidad de que existan transformaciones que requieran cierta intervención manual por parte del usuario.
- **Complejidad**. Las técnicas utilizadas dependen del nivel de complejidad de la transformación.
- **Preservación del significado**. Las transformaciones deben preservar ciertas características del modelo origen. Así, algunas deben preservar la estructura y otras el comportamiento.

Además, Mens clasifica las trasformaciones desde diferentes puntos de vista: según el lenguaje (metamodelo), dependiendo del nivel de abstracción (transformaciones verticales y horizontales), dependiendo de si cambia la sintaxis o también su semántica, o en función de si el lenguaje de los modelos origen y destino es el mismo (endógena) o distinto (exógena).

Pero quizás la clasificación más interesante en el proceso de desarrollo MDE es la realizada por Czarnecki [Czarnecki 03], que distingue dos tipos en base a la fase en que se realiza y al tipo de artefacto generado:

- Modelo a modelo (Model-To-Model (**M2M**)). Son las que tienen como salida otro modelo. Este tipo de transformaciones es el empleado para hacer evolucionar los modelos en el proceso de desarrollo. Czarnecki diferencia cuatro formas de realizar la transformación: manipulación directa de la representación de bajo nivel del modelo (posible con cualquier lenguaje), mediante la teoría de transformaciones de grafos, utilizando lenguajes declarativos, relaciones matemáticas o reglas de conversión y aproximaciones híbridas que combinan cualquiera de las anteriores.

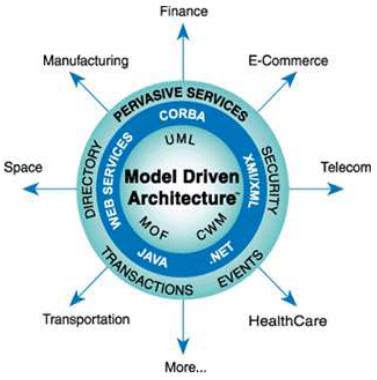
- Modelo a texto (Model-To-Text (**M2T**)). Es un caso particular de las anteriores para generar una representación textual del modelo origen sin metamodelo de destino. Se suelen utilizar para generar código en las etapas finales de desarrollo, aunque también se utilizan para otros fines, como generar documentación. Czarnecki distingue dos métodos para realizar estas transformaciones: utilizando el patrón *Visitor* [Gamma 95] para recorrer el modelo y generar texto, o utilizando soluciones basadas en plantillas, que mezclan el texto de la salida con comandos para recorrer el modelo.

La siguiente sección presenta una de las propuestas más extendidas en la línea de MDE.

3.4 MDA. La Propuesta MDE del OMG

MDA [Kleppe 03][MDA 03][Mellor 04] es el acrónimo de *Model Driven Architec*ture (Arquitectura Dirigida por Modelos). MDA es la propuesta del enfoque dirigido por modelos MDE del *Object Management Gr*oup (OMG), un consorcio de empresas (IBM, Borland, Hewlett-Packard o Boeing entre otras) que produce y mantiene una serie de especificaciones para permitir la interoperabilidad entre aplicaciones software.

Según la guía de MDA [MDA 03], MDA "*is an approach to using models in software development*". La principal característica diferenciadora de MDA respecto a los enfoques tradicionales para el desarrollo de software se encuentra en el uso de los modelos como el recurso principal en el proceso de desarrollo. MDA propone que los sistemas software sean generados directamente a partir de modelos de dicho sistema software.



MDA no pretende reemplazar paradigmas previos, lenguajes o herramientas, sino armonizarlos e integrarlos en base al enfoque dirigido por modelos. MDA organiza el desarrollo de software en tres capas denominadas M3 a M1, como se muestra en la Figura 3-11:

- La capa superior M3 contiene el metametamodelo reflexivo denominado *Meta-Object Facility* (MOF) [MOF 04]. MOF es común a todos los estándares OMG.
- En la capa M2 se encuentran los metamodelos (como UML [UML v2.1.1 07] o CWM [CWM 03]) que son utilizados por el usuario para crear los modelos que describen sistemas reales.
- La capa M1 alberga los modelos que representan sistemas reales.

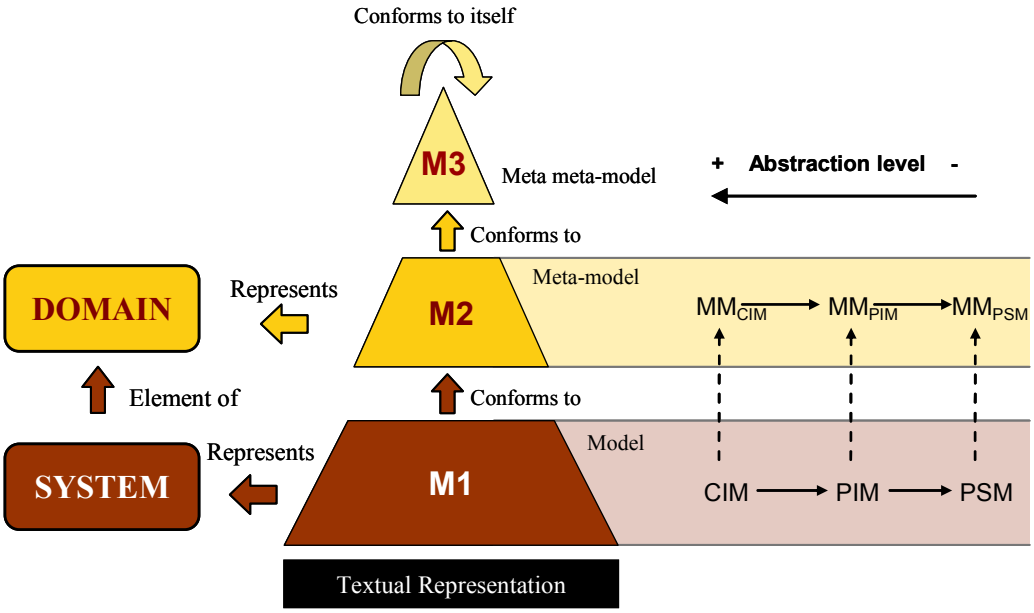


Figura 3-11. Capas en MDA.

La principal particularidad de MDA respecto a la filosofía genérica MDE es que organiza el desarrollo software de manera horizontal en tres familias de modelos en función del nivel de abstracción: CIM, PIM y PSM (véase la Figura 3-11):

- *Computer Independent Model* o modelo independiente de la computación (**CIM**). Se centra en el dominio del sistema así como en sus requisitos, detalles de la estructura y procesamiento. Esta forma de ver el sistema genera un Modelo Independiente de Computación (CIM – *Computation Independent Model*). Los CIMs se asocian a los modelos de dominio y es recomendable que sean especificados a partir de un vocabulario común para todos los elementos (computacionales o humanos) implicados en el desarrollo del sistema. El CIM trasciende a los sistemas. Cada proceso de negocio interactúa con trabajadores humanos y/o componentes de máquina. CIM describe solamente aquellas interacciones que tienen lugar entre los procesos y las responsabilidades de cada trabajador, sea o no humano.

  El objetivo fundamental del CIM es que cualquiera que pueda comprender el negocio y sus procesos pueda comprender un CIM, ya que éste evita todo tipo de conocimiento específico de la tecnología *hardware* o *software* en la que se vaya a implantar.

- *Platform Independent Model* o modelo independiente de la plataforma (**PIM**). Muestra la especificación del sistema teniendo en cuenta no sólo las especificaciones de funcionamiento propias – descritas en el CIM – sino también las especificaciones para la implementación en un medio informático. El PIM representa los aspectos que no cambiarán de una plataforma a otra, de acuerdo a una tecnología o método de implantación escogido, pero no muestra aspectos de implementación.

- *Platform Specific Model* o modelo dependiente de la plataforma (**PSM**). Muestra los detalles específicos de implementación, considerando el sistema operativo, la estructura de componentes, los lenguajes de programación utilizados, etc.

La transformación PIM a PSM no tiene por qué ser uno a uno, de hecho, una de las principales aportaciones de este enfoque es la posibilidad de realizar la implementación en distintas plataformas partiendo del mismo nivel PIM intermedio. Estas transformaciones se realizan mediante *modelos de marcas* para incluir la información adicional necesaria para poder realizar la transformación.

Cada nivel del enfoque MDA (CIM, PIM, PSM) involucra la definición de modelos conformes a sus respectivos metamodelos. A su vez, cada metamodelo estará definido conforme a un metametamodelo. Dado un metamodelo podremos definir un conjunto de modelos posibles a partir de las facilidades que nos ofrezcan los lenguajes definidos a tal efecto. Este esquema se repite en todos los niveles de MDA tal y como se detalla en la Figura 3-12. En cada nivel los modelos son conformes a sus respectivos metamodelos, y a su vez, dichos modelos pertenecen al conjunto de todos los posibles modelos (caja amarilla de la derecha en cada nivel) conformes a dicho metamodelo. Los metametamodelos (cajas azules a la izquierda) representan el conjunto de todos los metamodelos que se pueden definir conforme a ellos (cajas azules de la derecha). Las transformaciones necesarias entre los modelos definidos en el nivel CIM hacia modelos destino (flechas azules) en los niveles inferiores (PIM, PSM) vendrán dadas a partir de las respectivas correspondencias entre los conceptos que se manejan a nivel de metamodelo (flechas rojas).

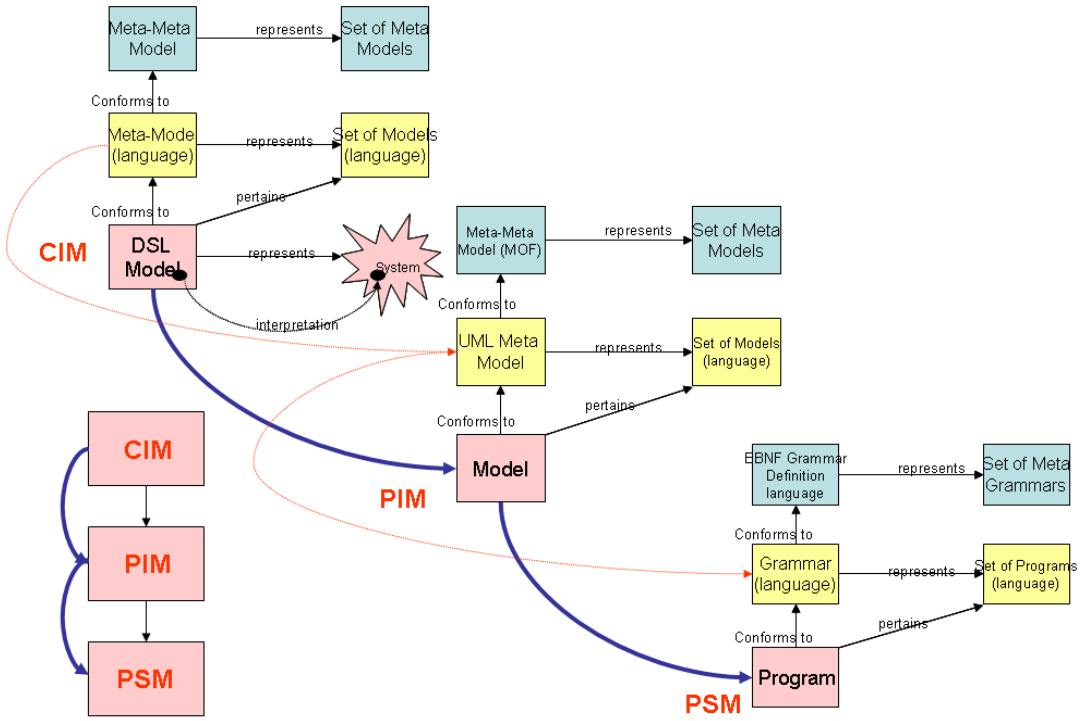


Figura 3-12. Niveles de modelado en las familias CIM, PIM y PSM.

Esta visión da un marco uniforme de gestión de modelos y metamodelos sin perder de vista las distintas perspectivas recomendadas por el enfoque MDA.

En esta línea OMG ha desarrollado un conjunto de estándares y metamodelos para dar soporte a la metodología MDA según esta organización de capas. Algunos de los más importantes se describen a continuación.

3.4.1 Meta-Object Facility (MOF)

Unos de los estándares más utilizados en MDA es *Meta Object Facility* (MOF). Se trata del estándar del OMG que define un lenguaje común y abstracto para definir lenguajes de modelado y cómo acceder e intercambiar modelos expresados en dichos lenguajes.

Como se observa en la Figura 3-11, MOF estaría situado en el nivel M3 del modelo de capas del OMG, el nivel de meta-metamodelo. Mediante MOF se puede definir cualquier lenguaje de modelado, incluido UML (véase la Figura 3-12).

La definición de MOF también permite la construcción de herramientas para manipular lenguajes de modelado, además de diversa funcionalidad adicional:

- *MOF Repository Interface*. Se trata de una interfaz que permite obtener información sobre modelos de nivel M1 de un repositorio basado en MOF.
- Intercambio de Modelos. MOF también define un formato de intercambio basado en XML para modelos, llamado XMI (*XML Metadata Interchange*). Ya que MOF se define usándose a sí mismo, puede usarse también XMI para generar formatos de intercambio de metamodelos.

El papel principal dentro de MDA es proporcionar los conceptos y herramientas para razonar sobre lenguajes de modelado. Gracias a ello se puede definir transformaciones sobre metamodelos de manera estándar.

3.4.2 XMI

XML Metada Interchange (XMI) [XMI 05] es un estándar del OMG que describe el formato de intercambio y almacenamiento persistente de modelos y metamodelos. XMI establece las correspondencias entre MOF y XML y define cómo deben ser usadas las etiquetas XML empleadas para representar modelos MOF serializados en XML.

Los metamodelos MOF son convertidos en definiciones de tipo de documento (DTD, *Document Type Definitions*) y los modelos son convertidos en documentos XML que son consistentes con su DTD correspondiente. XMI resuelve muchos de los problemas encontrados cuando intentamos utilizar un lenguaje basado en etiquetas para representar objetos y sus asociaciones y además, el hecho de que XMI esté basado en XML significa que tanto los metadatos (etiquetas) como las instancias que describen (elementos) pueden ser agrupados en el mismo documento, permitiendo a las aplicaciones entender rápidamente las instancias por medio de los metadatos.

Muchas de las herramientas CASE como Rational Rose, Together, Omondo, etc., soportan XMI y el estándar para importar y exportar en dicho formato. XMI no sólo puede ser usado como un formato de intercambio UML sino que puede ser utilizado para cualquier formato descrito por medio de un metamodelo MOF. Las herramientas compatibles con MOF permiten definir metamodelos o importarlos, por ejemplo de repositorios y herramientas CASE, y empezar a editar o modificar la información del modelo, por ejemplo instancias concretas de los servicios de negocios. Una vez hecho esto la información del modelo podría ser intercambiada, por medio de XMI, con otra herramienta compatible con MOF.

Un ejemplo existente hoy en día de un metamodelo compatible con MOF (además del conocido UML) es el CWM (*Commom Warehouse Metamodel*) [CWM 03], definido por el OMG. Éste es utilizado para describir metadatos para escenarios de *Datawarehousing*, e incluye elementos de modelo conocidos en el dominio de las bases de datos relacionales como tablas, columnas, vistas e índices.

XMI define muchos de los aspectos importantes involucrados en la descripción de objetos en XML:

- La representación de objetos en términos de elementos y atributos XML.
- Cómo los objetos están normalmente interconectados. XMI incluye un mecanismo estándar para enlazar objetos dentro del mismo fichero o entre fichero diferentes.
- La identidad de los objetos permite que éstos sean referenciados por otros objetos en términos de sus identificadores (*id* y *uuid*).
- El versionado de objetos y sus definiciones son gestionadas por el modelo XMI.
- La validación de documentos XMI se realiza por medio de sus DTD o XMI *Schemas*.

Además, XMI proporciona reglas para la creación de DTD, XML *Schemas* y documentos XML, y para ello define varios tipos de reglas de producción:

- Producción de DTD desde un modelo de objetos.
- Producción de XML *Schemas* desde un modelo de objetos.
- Producción de documentos XML desde un modelo de objetos.

3.4.3 OCL

El *Object Constraint Language* (OCL) [OCL 06] es un lenguaje propuesto para especificar las restricciones semánticas del diagrama de clases UML, junto a otro tipo de definiciones. OCL se puede utilizar en cualquier contexto en que se use el diagrama de clases al que esté asociado (como por ejemplo, en el modelo conceptual o en el diseño).

OCL es un lenguaje híbrido declarativo-imperativo, y es tipificado, porque usa el concepto de tipos de datos. Estos tipos de datos se encuentran jerarquizados a través de relaciones de inclusión. También existen varios tipos básicos predefinidos (similares a los de cualquier lenguaje) y unos pocos tipos paramétricos (o estructurados)

predefinidos. Los tipos no predefinidos van a estar asociados a las clases del diagrama de clases.

El componente central construido por este lenguaje es la expresión, que se manipula a través de operaciones que garantizan la ausencia de efectos colaterales. Éstas son características compartidas con los lenguajes funcionales, que son declarativos. Toda expresión válida para el lenguaje OCL debe tener asociado un tipo de datos. Las expresiones se pueden trasladar fácilmente al concepto de objetos, ya que cada tipo básico no predefinido se puede vincular a una clase del diagrama, y algunas de las operaciones de un tipo básico no predefinido se modelan como atributos y métodos/responsabilidades de la clase original. Las asociaciones entre clases también se modelan en OCL usando los roles de ambas clases participantes en la asociación (que deben estar rotulados en ambos extremos).

Las expresiones del lenguaje OCL se estructuran hasta llegar a su punto más alto, que es el de formar parte de una definición. En cada definición de OCL se describe cada restricción semántica del diagrama de clases, junto a otras propiedades también descriptas con expresiones. Cada definición en OCL se aplica a lo que se llama un contexto, que representa el destinatario del diagrama de clases sobre quien se aplica esa definición. OCL es un complemento ideal para MOF, ya que permite expresar restricciones semánticas a los metamodelos que MOF no puede añadir.

Además, OCL es un lenguaje de especificación con el que se pueden escribir expresiones sobre modelos, por ejemplo, el cuerpo de una operación de consulta, invariantes, pre y post-condiciones, etc. De esta manera se pueden definir modelos más precisos y completos. Algunos de los usos de OCL son:

- Especificar valores iniciales de atributos.
- Definir el cuerpo de operaciones de consulta.
- Establecer condiciones de guardia en diagramas de estados.
- Especificar las reglas de derivación de atributos o asociaciones.
- Expresar restricciones sobre clases o atributos.

Además de dar más precisión a los modelos, OCL puede usarse de manera muy efectiva en la definición de transformaciones: una primera expresión en OCL puede especificar los elementos en el modelo destino de la transformación.

3.4.4 QVT

Query, View and Transformation (QVT) [MOF-QVT 05] es el lenguaje propuesto por el OMG para definir transformaciones entre modelos.

QVT es un lenguaje declarativo-imperativo con el que se puede recorrer un modelo en busca de determinadas características o relaciones y definir transformaciones entre modelos conformes al mismo o a diferentes metamodelos. Con QVT se puede:

- Definir transformaciones uni y bidireccionales.

- Realizar actualizaciones incrementales entre modelos.
- Crear y destruir objetos del metamodelo.
- Definir transformaciones para comprobar si determinados modelos están relacionados.

3.5 Tecnología para Soportar MDE

Ninguna de las promesas de MDE (y en concreto MDA) sería hoy posible sin herramientas que den soporte a este nuevo enfoque. La mayor parte del peso de esta aproximación recae sobre estas herramientas de desarrollo de software que permiten realizar tareas de modelado, transformaciones de modelos, verificación de modelos, DSLs (Lenguajes Específicos de Dominio), generación de código, etc., por lo que el auge de MDA ha impulsado en los últimos años el desarrollo de tecnología (herramientas, lenguajes, etc.) tanto de carácter comercial como de propósito general que implementan total o parcialmente los conceptos de MDA y MDE.

Algunas herramientas que dan soporte a este enfoque son *Eclipse Modeling Framework (EMF)* [Steinberg 08][Stahl 06], *NetBeans MetaData Repository* [Matula 03], *OptimalJ* [Gil 05], *ArcStyler* [Arcstyler 05] y *AndroMDA*[Gil 05][AndroMDA 08]. *NetBeans MetaData Repository* da soporte tecnológico al estándar MOF constituyendo un repositorio de metainformación que puede ser modelada en UML. *OptimalJ* y *ArcStyler* son comerciales y *AndroMDA*, al igual que *ArcStyler* está basada en el concepto de cartucho como artefacto que engloba el lenguaje de modelado y las transformaciones para una determinada plataforma, existiendo en la actualidad cartuchos para la plataforma Java y para servicios *web*. En cambio, *Optimal-J* está dirigida a una arquitectura de tres capas con *Struts* (herramientas que dan soporte al desarrollo de aplicaciones *web*) *y EJB*.

Una aproximación similar es seguida en la computación integrada en modelos (*MIC – Model Integrated Computing* [Sprinkle 04]), con herramientas como *Generic Modeling Environment y MetaEdit+*. Todas estas herramientas constituyen un marco de metamodelado basadas en un lenguaje abstracto para definir la sintaxis, semántica y visualización de lenguajes específicos de dominio.

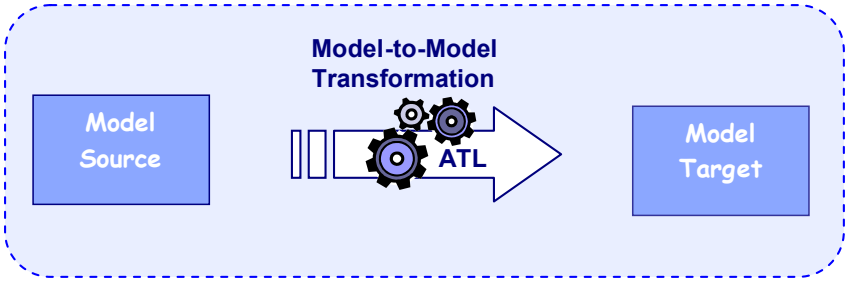
Sin embargo la elección de un entorno de desarrollo u otro limita enormemente el uso de otras herramientas MDA. Éste es el caso de las herramientas de modelado, donde las herramientas con más influencia en la actualidad son compatibles con múltiples entornos de desarrollo y muy especialmente el entorno de desarrollo Eclipse.

Por esta razón se ha decidido utilizar para este trabajo las herramientas presentadas a continuación:

- **Eclipse**: Es un entorno que puede ser utilizado no sólo en el desarrollo de productos software sino también en el desarrollo de herramientas que permitan construir productos software. Se puede decir que Eclipse es un marco de trabajo para el modelado y la integración de datos permitiendo almacenar metamodelos y

meta-datos. La principal ventaja de Eclipse radica en que su arquitectura está diseñada de forma que la mayoría de la funcionalidad proporcionada está localizada en *plugins* o en un conjunto de *plugins* relacionados. Esto hace a Eclipse una herramienta totalmente extensible.

- ***Eclipse Modeling Framework*** **(EMF**) es un entorno de modelado que proporciona facilidades para la generación de código. EMF puede ser utilizado como un marco de metamodelado donde el lenguaje común es un subconjunto de MOF. EMF se integra dentro del conjunto de *plugins* de Eclipse permitiendo tratar con gran variedad de artefactos software, como esquemas XML, modelos UML (definidos en entornos visuales de modelado como Rational Rose), esquemas relacionales (a través de Rational Rose), y ontologías, entre otros. Además, EMF es utilizada por las principales herramientas de IBM, aportando una visión industrial a nuestro enfoque de Gestión de Modelos.
- ***Graphical Modeling Framework*** **(GMF**) surge con el objetivo de generar automáticamente editores gráficos como plugins para Eclipse a partir de modelos, aprovechando la infraestructura ofrecida por EMF y GEF.
- ***Atlas Transformation Language*** **(ATL**) es una herramienta de transformación de modelos basada en los estándares del OMG, MOF, QVT y OCL 2.0. ATL forma parte del *framework* de gestión de modelos AMMA, que se encuentra integrado en Eclipse y EMF. Utiliza un lenguaje propietario llamado ATLAS para definir transformaciones que se ejecutan sobre JAVA y que proporciona un entorno de depuración. Es un lenguaje híbrido, ya que trabaja con construcciones declarativas e imperativas. Las construcciones declarativas son la opción preferida para escribir las transformaciones, puesto que son claras y precisas. Permiten expresar correspondencias entre los elementos del modelo fuente y destino, a partir de una serie de composiciones de reglas. Adicionalmente, las construcciones imperativas proporcionan constructores para facilitar la especificación de correspondencias que de forma declarativa serían mucho más complejas de implementar.



- ***MOFScript*****:** es una herramienta de transformación de modelo a texto presentada por el OMG. Presta particular atención a la manipulación de cadenas de caracteres y de texto y al control e impresión de salida de archivos. Es compatible con QVT y con MOF. Para las reglas de transformación, MOFScript define un metamodelo de entrada para la fuente (*source*) y sobre el cual operarán las reglas. El objetivo (*target*) es generalmente un archivo de texto (o salida de texto por pantalla).

- ***Java Emitter Template (JET)*** es un subproyecto de EMF centrado en simplificar el proceso de generación automática de código (Java, XML, JSP, etc.) a partir de plantillas. JET funciona a partir de plantillas que son traducidas a una clase Java para luego ser ejecutadas por medio de una clase generadora creada por el usuario.

Estas herramientas se sitúan en una posición ventajosa con el resto dado que presentan características tales como:

- **Similitudes con la arquitectura MDA**. Es un factor de relevancia que el entorno de desarrollo siga las directrices propuestas por el OMG en lo que a la arquitectura se refiere. Se tendrá en cuenta que se puedan diferenciar los niveles propuestos en MDA.
- **Multiplataforma**. Estas herramientas se integran en un entorno multiplataforma siguiendo las directrices del OMG. Esta es una característica muy importante, ya que permite utilizar las herramientas desarrolladas en case cualquier sistema operativo o procesador.
- **Compatibilidad con la plataforma Eclipse**. Todas estas herramientas se integran en forma de plugins en este entorno único.
- Posibilidad de **definición de transformaciones**.
- **Verificación de modelos**, para comprobar si un modelo se ajusta a las reglas de diseño o no.
- **Facilidad de comprensión de los modelos**. Los diagramas de los modelos y sus relaciones se presentan de forma visual y jerárquica y son fácilmente navegables.
- **Herramientas de soporte integradas**. Además del entorno de programación en sí, las herramientas ya vienen integradas con el entorno de desarrollo y se pueden integrar fácilmente herramientas nuevas.
- **Soporte para consistencia incremental**. Es una característica muy importante para el desarrollador. Tiene en cuenta la capacidad de conservación de variaciones en el código final tras la realización de nuevas transformaciones, lo que vulgarmente se podría denominar como "no machacar código".
- **Soporte del ciclo de vida de desarrollo**. Capacidad de integración dentro de un conjunto de herramientas que en conjunto cubran todas las fases del ciclo de vida de un producto *software*, donde el producto de una herramienta es directamente la entrada para otra.
- **Trazabilidad**. Capacidad de saber qué componente de un modelo inicial dio como resultado un componente de un modelo resultante tras una transformación.
- **Control y refinamiento de transformaciones**. Mide el grado de control y personalización de las transformaciones entre modelo.

3.6 Aplicación de MDE a Sistemas Reactivos

Los sistemas reactivos son sistemas heterogéneos cuyo rol fundamental es mantener una interacción continua con su entorno a través de dispositivos sensores y actuadores. Algunos ejemplos son los sistemas de inspección visual automatizados, las aplicaciones de redes de sensores inalámbricas, determinados sistemas robóticos y los sistemas domóticos. Para el desarrollo de estos sistemas se siguen tradicionalmente, como únicos criterios, la funcionalidad del sistema, la experiencia previa del diseñador, las limitaciones impuestas por ciertos requisitos no funcionales tales como el coste máximo asumible, etc. Pero quizás la limitación más importante de esta forma de proceder sea la dificultad de conseguir artefactos software reutilizables, prefiriendo, por regla general, una solución eficiente y totalmente a medida, antes que diseñar soluciones más generales para facilitar su reutilización. Como consecuencia de esto, cada nuevo sistema debe construirse prácticamente desde cero, aunque su lógica y estructura sean casi idénticas a las de otros desarrollados previamente pero implementados sobre plataformas diferentes.

La aplicación del enfoque dirigido por modelos MDE, y en concreto la propuesta MDA del OMG, permite resolver todos estos problemas asociados con el desarrollo tradicional de sistemas reactivos. En la Tabla 3-1 se recogen algunos de los problemas relacionados con este tipo de sistemas y las ventajas que supondría el empleo de MDA.

Problema Sistemas Reactivos	Implicaciones en el desarrollo	Aportaciones de MDA
Soluciones a medida de la plataforma utilizada en cada momento.	Aplicaciones poco flexibles y reutilizables.	Incremento de la flexibilidad y reutilización. Soluciones genéricas e independientes de la plataforma (niveles CIM y PIM).
Se requiere un alto grado de experiencia en la infraestructura desde las fases iniciales del diseño.	El proceso de diseño es lento y poco eficiente.	Aumento de la eficiencia en el proceso de diseño. En las fases iniciales del diseño se utilizan conceptos del dominio (lenguajes específicos de dominio), que son independientes de la plataforma.
Algunas de las herramientas utilizadas son muy dependientes de la infraestructura y su uso requiere amplios conocimientos de programación muy próximos a la plataforma de implementación.	El conocimiento en profundidad de una infraestructura es poco reutilizable en otras soluciones.	Herramientas genéricas. Soluciones reutilizables en distintas plataformas. La generación de código se realiza de forma (semi) automática en las fases finales del diseño (PSM).

Tabla 3-1. Problemas relativos al desarrollo de sistemas reactivos. Aportaciones de MDA.

Debido a la relativa juventud de MDA son escasas las publicaciones en las que se presentan estudios o resultados de su aplicación a sistemas reactivos. Una de las más interesantes es la recogida en [Alonso 08b][Vicente 07], que definen un metamodelo de nivel PIM y herramientas gráficas para dar soporte al desarrollo de diferentes sistemas reactivos (visión, robótica, redes de sensores, domótica, etc.), que permite la

descripción a nivel PIM de sistemas mediante tres vistas: arquitectural, comportamental y algorítmica. En la Figura 3-13 se muestra el ejemplo para la definición de la arquitectura de un robot guiado por un sistema de visión usando este entorno. Este trabajo ha sido utilizado para apoyar la propuesta para el desarrollo de sistemas domóticos realizada en esta Tesis, por lo que será revisado con más detalle en capítulos posteriores.

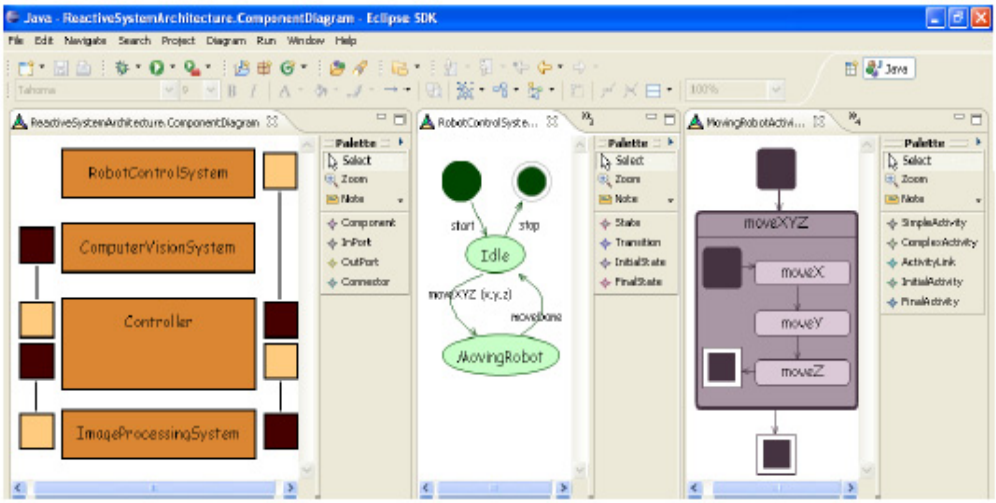


Figura 3-13. Herramienta V^{3}Studio para nivel PIM. Extraído de [Alonso 08b].

En el campo de las redes de sensores y actuadores (WSAN: *Wireless Sensors and Actuator Networks*), existen diversos trabajos que plantean el empleo de MDA y lenguajes específicos de dominio para todo el proceso de desarrollo [AlSaad 07]. En este campo es de interés el trabajo desarrollado en [Losilla 07a][Losilla 07b] una propuesta para redes de sensores similar a la que se plantea en esta Tesis para sistemas domóticos. En la Figura 3-14 se muestran las diferentes etapas del desarrollo WSAN-MDA; como punto de partida se utiliza un DSL específico para el dominio de redes de sensores (PIM) que permite definir los modelos o aplicaciones deseadas por el usuario. Mediante herramientas de transformación (M2M), el modelo de aplicación pretende ser transformado a un modelo de componentes específico de la plataforma (PSM), a partir del cual, mediante transformaciones modelo-a-texto (M2T) poder obtener de forma semiautomática el código ejecutable para los nodos de la red.

También existen algunas experiencias en robótica. En [Ortiz 07] se presenta la experiencia del empleo de una arquitectura basada en componentes para una familia de robots teleoperados y se propone la utilización de MDE para automatizar la generación de código específico para las diferentes plataformas a partir de componentes abstractos de la arquitectura desde el nivel PIM.

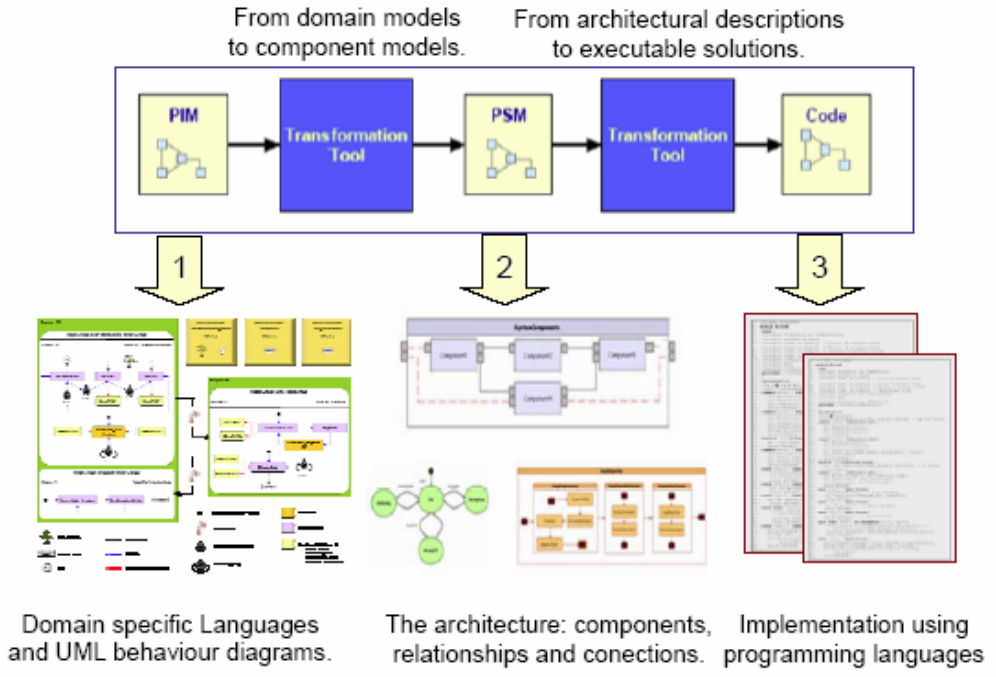


Figura 3-14. Aplicación de MDA al desarrollo de WSAN. Extraído de [Losilla 07b].

En el ámbito de la domótica, las referencias del empleo de MDA en el proceso de desarrollo son escasas. Markus Voelter, en su propuesta de desarrollo de software dirigido por modelos y aspectos para la implementación de líneas de producto [Voelter 07], presenta un caso de estudio para la familia de sistemas domóticos (véase la Figura 3-15), en el que se contempla la definición de un metamodelo a nivel PIM, relacionado con el dominio del problema, y otro a nivel PSM en el dominio de la solución. La evolución de los modelos se realiza mediante transformaciones modelo a modelo (M2M) y modelo a texto (M2T). En líneas generales, este trabajo es un excelente punto de partida para la definición de la arquitectura a utilizar para sistemas domóticos, con algunos matices:

1. Voelter trata el problema directamente en el nivel PIM. Sería conveniente definir el metamodelo y herramientas asociadas a sistemas domóticos en el nivel CIM e introducir un nivel PIM intermedio con la doble finalidad de disponer de un nivel intermedio de abstracción y, a su vez, permitir la integración a nivel de componentes con otros sistemas reactivos (redes de sensores, sistemas de visión, etc.).
2. El metamodelo para el dominio domótico ha de construirse para cada nueva aplicación. En uno de los ejemplos que plantea como caso de estudio contempla dispositivos de tres tipos: sensores, controladores y actuadores, y los elementos finales (luz, persiana, sensor de presencia, etc.) que se incluyen como clases estáticas en el metamodelo, por lo que **no es posible añadir nuevos dispositivos** una vez diseñada la arquitectura y las herramientas (¡para añadir un nuevo dispositivo habría que modificar el metamodelo!).

3. La generación de código se centra en la implementación OSGi [OSGi 08], que maneja servicios en Java, pero no se contempla la implementación en sistemas tan extendidos como KNX o Lonworks, en los que cada dispositivo contienen un procesador que ejecuta un código y se comunica a través de un bus de comunicación con el resto de nodos de la red.

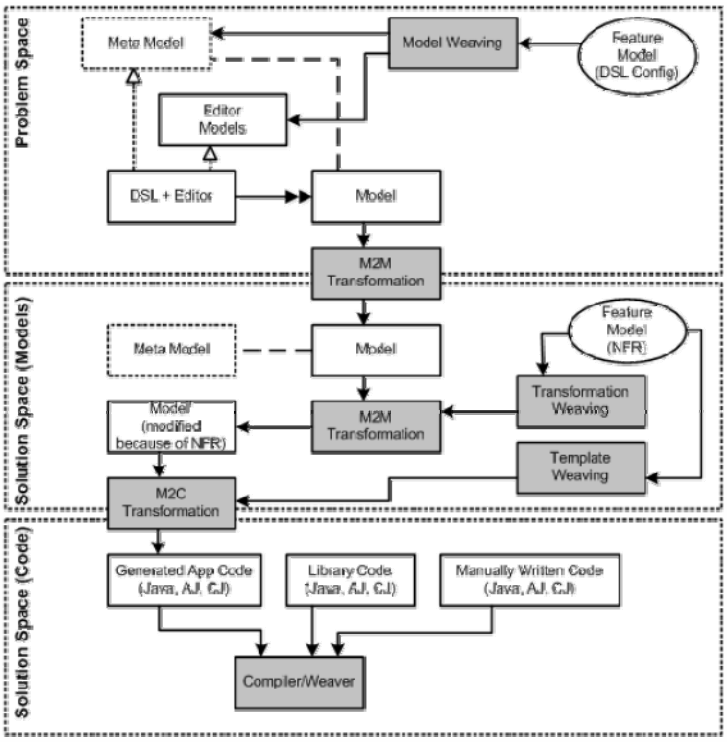


Figura 3-15. Propuesta de líneas de producto-MDA para sistemas domóticos [Voelter 07].

Otro planteamiento interesante es el de Muñoz et al [Muñoz 07][Cetina 07], que plantea también la necesidad de utilizar un enfoque dirigido por modelos en sistemas domóticos con el fin de aumentar el nivel de abstracción, la productividad y la calidad del software, además de mantener la independencia de la plataforma de implementación. Muñoz plantea las siguientes etapas en el diseño (véase la Figura 3-16):

1. Especificación de requisitos por el analista del sistema utilizando el lenguaje de modelado PervML. PervML consiste en una especificación UML del (1) modelo de servicios (diagrama de clases), (2) modelo de interacción (diagrama de interacción) y (3) modelo estructural (diagrama de componentes y servicios).
2. Un arquitecto del sistema (especialista en el *software* y el dominio de la tecnología domótica con la que se va a desarrollar la aplicación) selecciona los dispositivos y sistemas software a utilizar, generando la vista del arquitecto.
3. Un desarrollador implementa los *drivers* OSGi para gestionar los dispositivos o sistemas software.

4. Se aplica un motor de transformación y se generan automáticamente archivos Java y otros recursos auxiliares. Estos archivos se configuran para utilizar los *drivers* OSGi seleccionados y finalmente los archivos generados se compilan y empaquetan en archivos JAR para ser instalados en un servidor OSGi junto a los drivers.

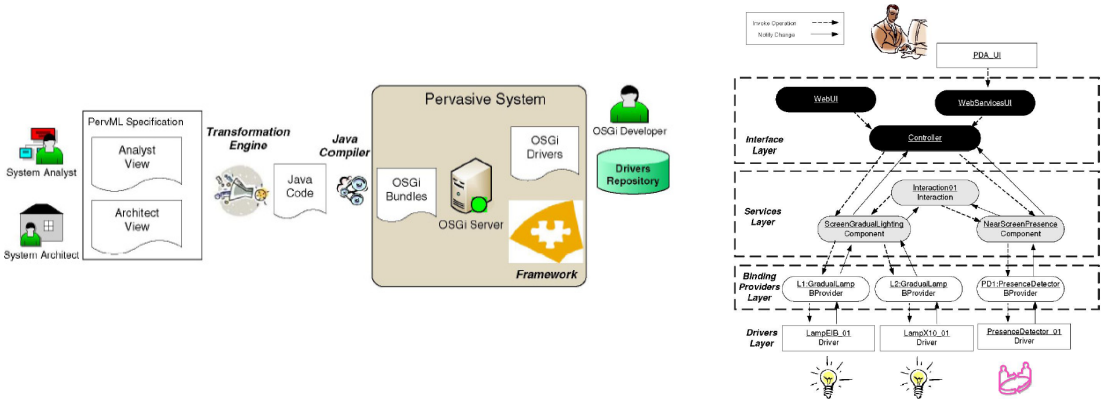


Figura 3-16. Metodología propuesta en [Muñoz 07][Cetina 07].

Siguiendo este planteamiento, Muñoz describe un caso de estudio para una sala de reuniones utilizando el sistema KNX/EIB como plataforma final de implementación [Muñoz 06].

Esta metodología también representa un buen ejemplo de las ventajas que supone el uso de MDA en el desarrollo de sistemas domóticos, pero presenta algunos inconvenientes:

- PervML usa UML en las vistas de analista y arquitecto. El lenguaje utilizado es poco intuitivo y poco cercano al espacio y términos del dominio domótico. Muñoz lo plantea como un lenguaje intuitivo **para desarrolladores de software,** pero no lo es para los especialistas del dominio, en su mayor parte ingenieros en disciplinas como la automatización o las telecomunicaciones.
- Utiliza *drivers* OSGi para controlar componentes domóticos por lo que, en el caso de plataformas finales como KNX/EIB, no aprovecha la capacidad de procesamiento de los dispositivos que podrían implementar toda la funcionalidad. El control se hace desde un ordenador o procesador central que implementa todo el código generado por los modelos a partir de la utilización de los drivers OSGi. En definitiva, se desaprovecha la filosofía de arquitectura distribuida de KNX/EIB o de Lonworks, convirtiendo a los dispositivos en meros sensores y actuadores controlados de forma centralizada.

Se trata, por tanto, de una metodología adecuada para el diseño interfaces de usuario para PDA, PC o WEB, pero no para diseñar un sistema domótico de principio a fin. Además, requiere de una intervención humana importante en las transformaciones de modelo a modelo y de modelo a código.

3.7 Conclusiones y Aportaciones a la Tesis

Hoy día, el desarrollo de sistemas reactivos, y en concreto de los sistemas domóticos (que entrarían también dentro de la categoría de sistemas pervasivos o de computación ubicua) es un campo de intensa investigación. En el diseño de este tipo de sistemas se han utilizado tradicionalmente tecnologías de bajo nivel de abstracción y *frameworks* de implementación específicos del dominio, lo que lleva a una escasa reutilización y eleva las posibilidades de errores en el diseño. Por ello, la utilización de nuevas técnicas de la Ingeniería del *Software* se plantea como una necesidad en este campo.

La metodología de desarrollo dirigido por modelos (MDE) es una tecnología relativamente reciente (se empieza a plantear en el año 2000), que se ha visto impulsada en los últimos años por la aportación del OMG con su propuesta MDA y los numerosos estándares y herramientas que están haciendo posible su aplicación en la actualidad.

En este capítulo se han realizado aportaciones fundamentales para el desarrollo de esta Tesis Doctoral. En primer lugar se han planteado los principios básicos del modelado, lo que permite elevar el nivel de abstracción y solucionar una de las carencias más importantes en el desarrollo tradicional de sistemas reactivos. Además se han descrito los beneficios que MDE y la propuesta MDA pueden aportar al diseño software. Algunos de los aspectos más importantes son la utilización de conceptos del dominio de manera independiente de la plataforma, la evolución de los modelos mediante transformaciones desde las especificaciones iniciales del problema hasta la generación de código ejecutable, y las mejoras en flexibilidad, reutilización y calidad de los diseños realizados.

Otra aportación importante es la descripción de las herramientas que se utilizarán para dar soporte a la propuesta de diseño de sistemas domóticos dirigido por modelos.

Finalmente se han revisado los trabajos existentes en el ámbito de los sistemas reactivos que utilizan el enfoque MDA. Este estudio ha permitido establecer algunos de los fundamentos empleados en este trabajo de Tesis y servirán de precedente para un conocimiento detallado del trabajo realizado.

CAPITULO 4

Lenguaje Específico de Dominio

Este capítulo presenta un lenguaje específico para el dominio de la domótica. Para ello es necesario un estudio detallado de los aspectos más importantes involucrados en el diseño de este tipo de DSLs.

Con esta filosofía se pretenden solucionar los problemas asociados al proceso de diseño que se sigue actualmente en el ámbito de la domótica. Entre otros, agilizar el proceso desarrollo de aplicaciones, facilitar la verificación y aumentar el nivel de abstracción en la definición de los requisitos del sistema. El lenguaje específico de dominio (DSL) para el modelado recogerá los requisitos de aplicación de forma gráfica e intuitiva para el desarrollador. Siguiendo la filosofía MDE, dicho DSL se ha definido conforme a un metamodelo que especifica las relaciones entre los conceptos del domino.

El capítulo se organiza en siete apartados. El primero revisa la literatura existente en el campo de los lenguajes específicos de dominio, identificando las ventajas e inconvenientes de su uso así como los enfoques que se pueden utilizar a la hora de abordar su diseño para un campo de aplicación concreto. El siguiente expone los elementos que conforman un lenguaje y algunas directrices para su diseño. El apartado tres revisa los escasos DSL existentes para el dominio de la domótica indicando sus cualidades y deficiencias. A continuación, el apartado cuatro describe el DSL propuesto para desarrollar aplicaciones domóticas y el metamodelo que le da soporte. Finalmente, las últimas secciones presentan las herramientas utilizadas para crear el DSL en un marco de gestión de modelos basado en la propuesta MDA, el enfoque utilizado para abordar la definición de la semántica y, por último, las conclusiones que pueden extraerse.

4.1 Lenguajes Específicos de Dominio

En todas las ramas de la ciencia y de la ingeniería se pueden distinguir dos tipos de enfoques a la hora de abordar un problema: genéricos y específicos. Los enfoques genéricos proporcionan una solución general para muchos problemas en un determinado área, pero la solución no suele ser óptima. Un enfoque específico ofrece una solución mucho mejor para un conjunto más reducido de problemas. Esta dicotomía se presenta en numerosas publicaciones de Ingeniería del *Software* en términos de *lenguajes específicos de dominio (DSLs) vs. lenguajes de programación genéricos (GPLs)*.

Los DSLs son lenguajes de programación o especificaciones, a veces ejecutables, que ofrecen, gracias a notaciones y abstracciones adecuadas, un gran poder expresivo centrado, y generalmente restringido, a un dominio concreto. Arie van Deursen revisa en [Deursen 00] setenta y ocho publicaciones relativas a lenguajes específicos de dominio. El interés de este tipo de lenguajes en Ingeniería del *Software* es tal que se han publicado números especiales de varias revistas dedicados a este tema [Mernik 01][Mernik 02][Wile 99].

Durante años, los DSLs se han diseñado y utilizado ampliamente para propósitos muy diversos. En la Tabla 4-1 se muestran algunos DSLs de uso habitual en diferentes dominios. A pesar de ello, su estudio sistemático ha comenzado recientemente, impulsado por el planteamiento de la metodología de desarrollo de software dirigido por modelos (MDE). Los DSLs se utilizan en el contexto MDE para guiar las diferentes etapas de diseño y además, el enfoque MDE ayuda a reducir el coste de desarrollo de los DSLs. Se trata, por tanto, de una unión sinérgica que permite mejorar significativamente el proceso de desarrollo del software.

DSL	Dominio de aplicación
BNF	Especificación de sintaxis
Lenguaje de macros de Excel	Hojas de cálculo
HTML	Páginas de hipertexto
LATEX	Edición de textos
Make	Generación de código ejecutable
SQL	Consultas de bases de datos
VHDL	Diseño hardware

Tabla 4-1. Lenguajes específicos de dominio de uso habitual [Mernik 05].

Numerosos autores analizan las ventajas e inconvenientes del desarrollo de DSLs frente al uso de lenguajes de propósito general. A continuación se enumeran las más importantes, extraídas y resumidas de [Deursen 00][Estublier 05] y [Czarnecki 05].

Entre las ventajas cabe destacar que:

- Los DSLs permiten expresar soluciones en el idioma usando únicamente conceptos y reglas del dominio del problema. En consecuencia elevan el nivel

de abstracción. Por lo tanto, los expertos del dominio pueden entender, validar, modificar e incluso desarrollar DSLs por sí mismos.

- Los programas son concisos, en gran medida autoexplicativos y pueden ser reutilizados para diferentes propósitos.
- Proporcionan una notación (que puede ser gráfica) cercana a la forma natural de pensar del experto del dominio, evitando la confusión sintáctica que se produce al utilizar lenguajes de propósito general.
- Mejoran la fiabilidad, mantenimiento y portabilidad.
- Posibilitan la automatización (parcial o total) del proceso de desarrollo, incrementando la productividad.
- Contienen el conocimiento del dominio y permiten su conservación y reutilización.
- Mejoran la verificación al facilitar la construcción de analizadores estáticos para detectar más errores que los analizadores para lenguajes de propósito general, al tiempo que permiten describir los errores en un lenguaje más cercano al experto del dominio.
- Permiten generar código optimizado basándose en el conocimiento del dominio, que normalmente no está disponible para los compiladores de lenguajes de propósito general.

Y entre los inconvenientes de la utilización de DSLs cabe reseñar los siguientes:

- Aumenta el coste del diseño, implementación y mantenimiento del lenguaje.
- Aumenta el coste de formación de los usuarios.
- En la actualidad existen pocos DSLs.
- Dificultad de encontrar un ámbito adecuado que justifique su desarrollo.
- Dificultad para encontrar el equilibrio entre los constructores específicos del dominio y los de propósito general.
- Pérdida potencial de eficiencia frente al software desarrollado "a mano".

Muchos de los inconvenientes descritos en la literatura son discutibles o se pueden evitar. En lo que ser refiere al coste, el DSL será viable económicamente sólo si se van a desarrollar muchas aplicaciones en el dominio para el que se ha diseñado o bien se dispone de herramientas que reduzcan el coste de su desarrollo. En los últimos años es cada vez mayor el número de herramientas especializadas, como *MetaEdit+*[1], *Microsoft DSL Tools*[2], *Telelogic Rhapsody*[3] de IBM (comerciales), o *Eclipse Modeling Project*[4] (libre).

[1] http://www.metacase.com
[2] http://msdn2.microsoft.com/en-us/library/bb126235.aspx
[3] http://modeling.telelogic.com/products/rhapsody/index.cfm
[4] http://www.eclipse.org

Esta última herramienta proporciona un conjunto de *plugins* (como EMF, GMF, ATL y QVT) para dar soporte a todo el ciclo de desarrollo basado en modelos, y es la que está evolucionando a mayor velocidad, puesto que al tratarse de un proyecto libre y extensible recibe continuamente aportaciones de la comunidad científica.

Por otro lado, el coste de formación de usuarios está estrechamente relacionado con los conceptos utilizados en el diseño del DSL: cuanto más cercanos sean al dominio, más facilitarán su comprensión por usuarios habituados a ese dominio (una de las grandes ventajas de los DSLs), por lo tanto, en la mayoría de los casos la afirmación relativa al coste de formación de los usuarios [Deursen 00] no es cierta. Además, las propuestas de DSLs desde la aparición de la metodología MDE son cada vez más numerosas. Finalmente, la discusión sobre la eficiencia es similar a la planteada cuando se crearon los primeros compiladores que evitaban la utilización directa de código en ensamblador; la eficiencia depende mucho de las herramientas y metodología utilizada, y se compensa sobradamente con la mejora en la productividad, reutilización y calidad del software.

Por lo tanto, antes de abordar el diseño de un DSL para una aplicación, hay que tener en cuenta todas estas ventajas e inconvenientes. En esta línea, D. Spinellis propone en [Spinellis 01] una serie de patrones para ayudar al diseño de este tipo de lenguajes. A su vez, M. Mernik presenta en [Mernik 05] un método para determinar cuándo y cómo desarrollar un DSL. Asimismo, J. P. Tolvanen expone en [Tolvanen 06] los pasos para iniciar el diseño de un DSL y algunos ejemplos ilustrativos. Pero quizás una de las aportaciones más interesantes es la presentada por J. Luoma [Luoma 04], donde se realiza un estudio exhaustivo de veintitrés DSLs desarrollados en diferentes dominios para automatizar el desarrollo de software basado en modelos siguiendo los principios de MDE. J. Luoma identifica cuatro categorías o enfoques no excluyentes entre sí como conductores de la identificación de los constructores del lenguaje:

c1. Conceptos del desarrollador o experto del dominio.

c2. Generación de la salida.

c3. Interfaz o aspecto final del sistema construido.

c4. Espacio de variabilidad.

Un gran número de DSLs se basan en los **conceptos utilizados por los expertos del dominio** (c1), son fáciles de definir y pueden ser verificados por los propios expertos. La principal ventaja de este enfoque es que eleva el nivel de abstracción más allá de los conceptos de programación, por lo que la plataforma final de implementación se puede cambiar sin realizar modificaciones en el DSL (utilización de distintos PSM a partir de modelos CIM y PIM comunes). Esto facilita en gran medida su utilización por el usuario final, que no tiene por qué tener conocimientos de programación. La Figura 4-1 (c1) ilustra un DSL creado con este método para modelar seguros y productos financieros. Como se puede apreciar, tanto los conceptos como las relaciones entre ellos se representan mediante constructores gráficos de fácil identificación por el usuario.

Otra familia de DSLs es la de aquellos generados a partir de los **constructores de código de la salida** que se genera (véase la Figura 4-1 (c2)). Tal es el caso de lenguajes como CPL (*Call Processing Language* [Lennox 04]), *plugins* añadidos a algunos entornos de programación, o lenguajes de diseño de esquemas, en los que los constructores del DSL se extraen directamente de los constructores del lenguaje que se quiere generar. En estos casos el nivel de abstracción no se eleva mucho, y tampoco se consigue una independencia de la implementación final, aunque se mejora la productividad al facilitar el proceso de diseño y la detección temprana de errores.

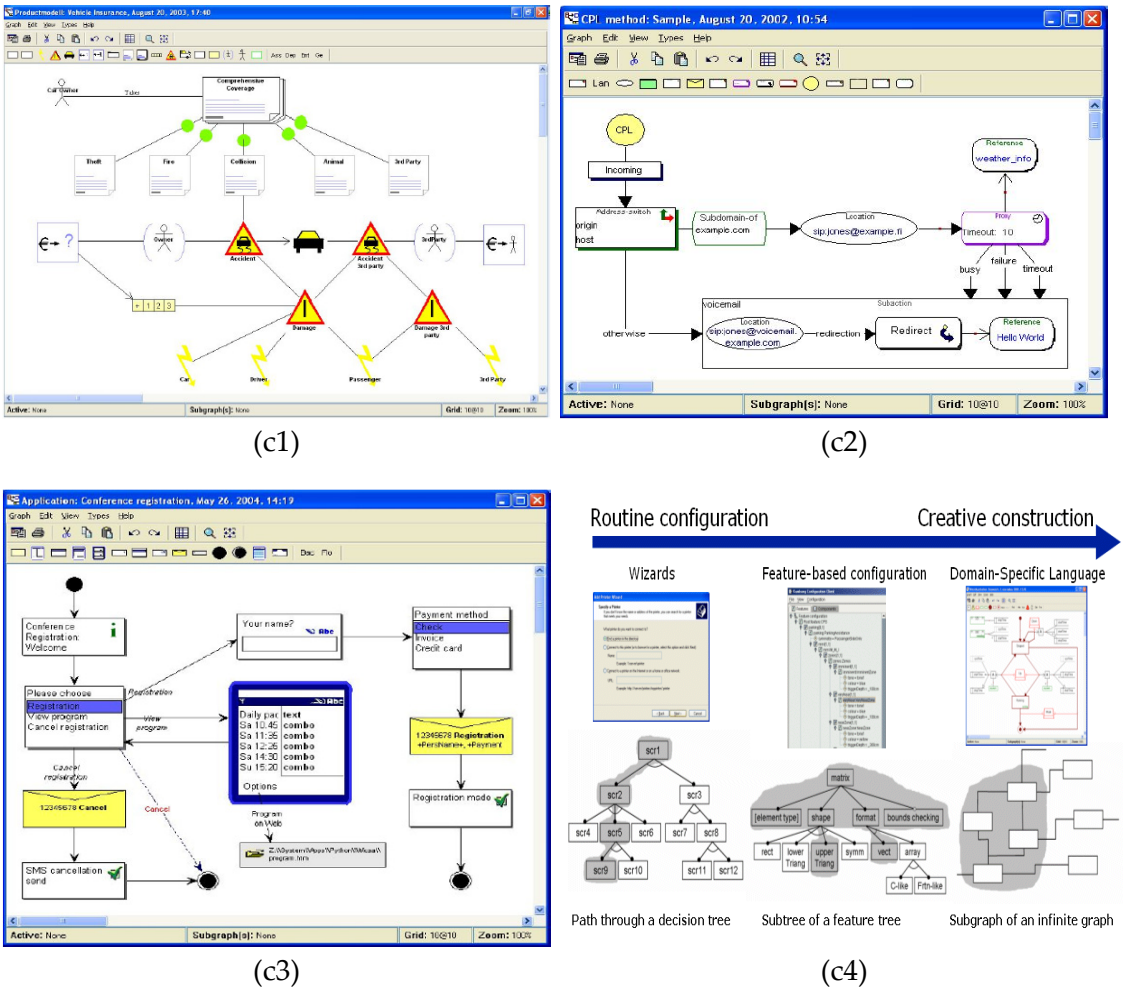


Figura 4-1. Ejemplos de DSL desarrollados según los cuatro enfoques. (c1) Conceptos del dominio: modelado de seguros. (c2) Generación de la salida: procesamiento de llamadas. (c3) Interfaz del sistema: DSL para desarrollo de interfaz de teléfonos móviles. (c4) Espacio de variabilidad: rangos de decisiones y características incluidos en la sintaxis del lenguaje. Extraídos de [Tolvanen 06] [Luoma 04].

El tercer enfoque se basa en utilizar conceptos de la **interfaz final del sistema** tal y como la va a percibir el usuario (véase la Figura 4-1 (c3)). Con este planteamiento se puede ahorrar mucho esfuerzo en el modelado, ya que la interfaz recoge una gran cantidad de información específica del dominio. La definición del lenguaje es bastante directa. El reto se plantea a la hora de definir otros elementos y restricciones. Si los constructores de la interfaz son lo suficientemente ricos como para cubrir la

funcionalidad se aumenta mucho el nivel de abstracción. Existen numerosos ejemplos basados en este enfoque, como el desarrollado por Nokia para sus teléfonos móviles [Nokia 03]. En este caso, existía previamente una arquitectura de soporte y una API (*Application Programming Interface*) bastante sólida a partir de la cual se creó el DSL.

Finalmente, existen DSLs que se definen a partir de un estudio de la **variabilidad** del domino (véase la Figura 4-1 (c4)). Son típicos en las familias de productos y sus constructores se obtienen analizando las características comunes y diferencias de los productos que se desean obtener. El nivel de abstracción que se logra depende de la naturaleza de la variabilidad: en aquellos dominios en que la variabilidad futura es fácil de predecir el nivel de abstracción es mayor que en aquellos en que resulta difícil. El diseño basado en variabilidad es bastante complejo, y casi nunca se basa únicamente en este enfoque, sino que se combina con alguno de los anteriores. Estos DSLs son los que con más frecuencia se utilizan en el enfoque de líneas de producto, que se basa en el mismo principio de análisis de variabilidad.

De este estudio se pueden extraer varias conclusiones interesantes que han servido de ayuda para definir el DSL propuesto en este trabajo de Tesis:

- La utilización de un DSL permite elevar el nivel de abstracción, aunque no todos los enfoques lo consiguen en la misma medida.
- La definición de todos los lenguajes estudiados en diferentes dominios está basada en metamodelos, lo que da total libertad para identificar los conceptos básicos del lenguaje.
- Ninguno realiza la definición extendiendo conceptos de UML con técnicas como el uso de perfiles.
- Los enfoques no son ortogonales, en muchos casos se utilizan varios simultáneamente, de manera que se aprovechan las ventajas de cada uno de ellos.

J. P. Tolvanen propone también las siguientes etapas para la definición de un DSL [Tolvanen 06]:

1. Identificar las abstracciones y cómo relacionarlas.
2. Especificar los conceptos y reglas del lenguaje (metamodelo).
3. Crear la representación visual del lenguaje (notación).
4. Definir las herramientas para comprobación de modelos, transformaciones, documentación, generación de código, etc.

En el siguiente apartado se exponen con más detalle los elementos propios de un lenguaje y las etapas necesarias para su diseño.

4.2 Elementos que Conforman la Definición de un Lenguaje

Todos los lenguajes, ya sean específicos de un dominio, de propósito general, de programación o de modelado, comparten unas características cuya comprensión es básica para su desarrollo [Clark 08]:

- **Sintaxis concreta**. Es la notación que facilita la presentación y construcción de modelos y programas en el lenguaje. Puede ser de dos tipos:
    - Sintaxis textual. Permite escribir los modelos o programas de modo textual. Típicamente consiste en declaraciones de objetos, variables y expresiones que establecen propiedades relacionadas con dichos objetos y variables (por ejemplo código en Java). Tiene la ventaja de permitir capturar expresiones complejas, pero a partir de cierto número de líneas el texto es difícil de comprender y manejar.
    - Sintaxis gráfica. Representa el modelo o programa en forma de diagramas basados en iconos que representan vistas del modelo subyacente. Permite expresar muchos detalles de forma intuitiva y comprensible. El único inconveniente surge si se pretende expresar un gran nivel de detalle, en cuyo caso puede resultar complejo y difícil de entender.

  En la práctica se puede utilizar una combinación de las dos: notación gráfica para las vistas de más alto nivel del modelo con sintaxis textual para las propiedades detalladas.

- **Sintaxis abstracta**. Describe el vocabulario de conceptos proporcionados por el lenguaje y cómo se pueden combinar para crear modelos. Consiste en una división de conceptos, las relaciones que existen entre ellos y reglas que establecen cómo se pueden combinar los conceptos de manera correcta. Por ejemplo, en una máquina de estados el modelo de sintaxis abstracta incluye conceptos como Estado, Transición y Evento, las relaciones son del tipo "las transiciones están ligadas a un estado origen y un estado destino", y las reglas "dos transiciones no pueden dispararse por el mismo evento". La sintaxis abstracta del lenguaje es independiente de su sintaxis concreta y su semántica (que se expone seguidamente), sólo trata con la forma y estructura de conceptos en un lenguaje sin tener en cuenta su presentación o significado.

- **Semántica**. La sintaxis abstracta contiene poca información del significado del lenguaje por lo que es necesaria información adicional para capturar su semántica y para tener claro lo que el lenguaje representa y significa (por ejemplo, en una máquina de estados hay que definir qué pasa si dos transiciones salen del mismo estado, qué es un estado, etc.). Asimismo, la semántica se debe definir de manera

precisa para que el usuario entienda el lenguaje, ya que representará la interpretación que se haga del lenguaje en una máquina de ejecución (virtual o no).

Además de estas características, deben existir mecanismos que permitan transformar o relacionar el lenguaje con otros lenguajes, lo que se consigue mediante transformaciones en el enfoque MDA. El lenguaje debe ser también extensible y adaptable para soportar su evolución y adaptabilidad a nuevos requerimientos.

4.2.1 Lenguajes y Metamodelos

El uso de metamodelos en la definición de estas características es esencial ya que permite realizar las definiciones de forma unificada y aportando riqueza semántica. Por lo tanto, la aplicación de MDA es sinérgica con la definición del DSL, proceso en el que se utilizará este enfoque. Así, mediante metamodelos, se puede describir la sintaxis concreta, la abstracta y la semántica de un lenguaje.

Es fundamental contar con herramientas que den soporte al metamodelado. En el apartado "4.5 Soporte al DSL en un Marco de Gestión de Modelos" se proporciona una visión de las herramientas utilizadas en este trabajo de Tesis. Dichas herramientas permiten el desarrollo de un lenguaje para el dominio domótico en el contexto del enfoque MDA propuesto por el OMG que se presentó en el capítulo anterior.

La tarea de crear el metamodelo de un lenguaje no es trivial, y será tanto más compleja cuanto mayor sea la complejidad del lenguaje por definir. Pero hay un proceso iterativo y bien definido para ello:

1. Definir la sintaxis abstracta.
2. Definir las reglas para asegurar la corrección de las expresiones derivadas de la sintaxis abstracta.
3. Definir la sintaxis concreta.
4. Definir la semántica.
5. Construir correspondencias con otros lenguajes o modelos si se precisa.

T. Clark propone cinco niveles para evaluar la calidad de un metamodelo para un lenguaje [Clark 08], que se presentan resumidos en la Tabla 4-2. La mayoría de los metamodelos existentes no pasan de nivel 2. UML no pasa del 3. El objetivo de un lenguaje totalmente definido debería llegar al nivel 5.

Nivel 1
• Se ha definido un modelo de sintaxis abstracta pero no se ha probado en una herramienta. • La semántica del lenguaje que define es informal e incompleta. • Hay pocas o ninguna regla.
Nivel 2
• El modelo de sintaxis abstracta está relativamente completo. • Se ha definido un número importante de reglas. • Todo o parte del modelo se ha comprobado en una herramienta. • Se han realizado ejemplos partiendo de la sintaxis abstracta y se han utilizado para validar su corrección. • La semántica está todavía definida de manera informal, pero se está en el camino de analizar la semántica del lenguaje.

Nivel 3
• La sintaxis abstracta del modelo está completamente probada y verificada. • Se ha definido la sintaxis concreta del lenguaje, pero está sólo parcialmente formalizada. • Típicamente, la sintaxis concreta se describe en términos de ejemplos informales más que en un modelo preciso. • Se han hecho algunas consideraciones respecto a la extensibilidad de la arquitectura del lenguaje, pero no se han formalizado o comprobado.
Nivel 4
• La sintaxis concreta del lenguaje se ha formalizado y comprobado. • Los usuarios pueden crear modelos visuales o textuales y comprobar que se generan instancias válidas del modelo de sintaxis abstracta. • La arquitectura del lenguaje se ha modificado para facilitar la reutilización y extensibilidad. • Empiezan a aparecer modelos de la semántica.
Nivel 5
• Se han modelado todos los aspectos del lenguaje, incluyendo su semántica. • El modelo semántico es ejecutable, permitiendo a los usuarios del lenguaje realizar operaciones semánticamente ricas en modelos escritos en el lenguaje, como simulación, evaluación y ejecución. • La arquitectura del lenguaje soporta buenos niveles de reutilización, lo que se ha demostrado mediante ejemplos reales. • En el caso óptimo, el metamodelo completo no se basa en ninguna tecnología externa, es decir, es una definición del lenguaje totalmente independiente de la plataforma y autocontenida, que podrá ser utilizada "como es" para generar o instanciar herramientas.

Tabla 4-2. Niveles para evaluar la calidad del metamodelo de un lenguaje [Clark 08].

A continuación se describen con más detalle las características de las sintaxis concreta, abstracta y la semántica de un lenguaje y cómo construirlas empleando metamodelos.

4.2.2 Sintaxis Abstracta

Construir el modelo de la sintaxis abstracta es el primer paso en el diseño de un lenguaje de modelado, ya que describe los conceptos del lenguaje y las relaciones entre ellos y define las reglas que determinan si los modelos escritos en el lenguaje son o no válidos. Los conceptos, relaciones y reglas identificados en este paso proporcionarán un vocabulario y una gramática para construir modelos en el lenguaje.

En este contexto se entiende como **concepto** cualquier cosa que representa una parte del vocabulario del lenguaje. Por lo tanto el modelado se centra en la **representación abstracta de los conceptos**, más que en la concreta, y en las relaciones estructurales que existen entre los conceptos del lenguaje (y nada en la semántica).

Debe contener **reglas** para determinar si los modelos escritos están bien formados (son sintácticamente válidos). Las reglas son muy útiles a la hora de implementar la herramienta para dar soporte a lenguaje.

Los modelos de sintaxis abstracta se escriben en un **lenguaje de metamodelado.** En este trabajo de Tesis se utiliza EMF (*Eclipse Modeling Framework*) que proporciona:

- Clases para describir los conceptos del lenguaje.
- Paquetes para particionar el modelo.
- Atributos y asociaciones para describir relaciones entre conceptos.
- Restricciones para expresar reglas (expresadas en lenguaje OCL).

Los pasos implicados en el desarrollo del modelo de sintaxis abstracta son los siguientes:

1. Identificación de conceptos: utilizar cualquier información disponible para identificar los conceptos que usa el lenguaje y reglas obvias que determinen la validez de los modelos. Algunas técnicas aplicables son:
   - Construir una lista de conceptos candidatos del lenguaje centrándose en determinar los que tienen sentido. Se pueden seguir los siguientes criterios:
     - Conceptos que tienen nombres.
     - Conceptos que contienen otros conceptos (por ejemplo, una clase contiene atributos).
     - Conceptos que reflejan información sobre relaciones con otros conceptos.
     - Conceptos que desempeñan el papel de espacios de nombres para conceptos nombrados.
     - Conceptos que muestran una relación tipo/instancia.
     - Conceptos que se descomponen recursivamente.
     - Conceptos que son parte de una expresión o están asociados con expresiones.
   - Construir ejemplos de modelos usando el lenguaje:
     - Utilizar cualquier notación que parezca adecuada para representar los conceptos y construir modelos de ejemplos reales.
     - Si hay lenguajes preexistentes habrá recursos disponibles como definiciones BNF de la sintaxis que se pueden trasladar al metamodelo y ejemplos de uso.

   Una vez que se tienen los ejemplos, habrá que abstraerse de ellos para identificar conceptos genéricos del lenguaje y las relaciones entre ellos. Es útil anotar los ejemplos con conceptos de modelado como paso previo a dicho modelado. Los ejemplos de modelos no válidos pueden servir para identificar las reglas.

   En general, los modelos más simples son los mejores. La complejidad debida a la representación con diagramas se debe dejar para los modelos de sintaxis concreta.

2. Casos de Uso. Una buena técnica para identificar conceptos es considerar casos de uso asociados con el uso UML del lenguaje. Es lo más semejante a escribir una interfaz con el metamodelo.

3. Modelado de Conceptos. Una vez identificados los conceptos se pueden usar características propias de modelado de la orientación a objetos (se pueden encontrar algunos ejemplos en [Larman 02]):
   - Los conceptos se describen con clases, con atributos que capturan sus propiedades.
   - Las relaciones entre conceptos se describen con asociaciones.
   - Cuando tiene sentido definir categorías de conceptos se utiliza generalización.

   Una estrategia útil es reutilizar definiciones existentes de lenguajes, si es posible.

4. Reglas. Una vez construido el modelo básico se deben identificar ejemplos de modelos correctos e incorrectos que se pueden escribir para así definir las reglas.

Hay que intentar que sean lo más generales posible e investigar conflictos que puedan existir entre dichas reglas [Warmer 99].

5. Validación y Comprobación. Es muy importante validar la corrección del modelo de sintaxis abstracta (ahorra esfuerzos en etapas posteriores del ciclo de vida). Una técnica útil consiste en construir instancias del modelo de sintaxis abstracta que se correspondan con modelos ejemplo (un diagrama de objetos ayuda bastante). La mejor manera de comprobar la corrección del lenguaje es construir una herramienta que lo implemente, de manera que lo puedan probar los usuarios finales. Hay arquitecturas (o herramientas como Eclipse) que facilitan la generación rápida de herramientas a partir de metamodelos.

Se ha descrito el proceso y el lenguaje de modelado para modelar la sintaxis abstracta de lenguajes, obteniendo como resultado la definición de los conceptos del lenguaje y las relaciones que lo gobiernan. Para la definición completa del lenguaje queda por definir aún la sintaxis concreta y la semántica.

4.2.3 Sintaxis Concreta

Como se ha demostrado, la sintaxis abstracta define el vocabulario subyacente y la gramática de un lenguaje, pero no define cómo se presenta al usuario. Esto es lo que hace la sintaxis concreta de manera textual o gráfica. Crear la sintaxis concreta supone dos etapas: primero, interpretar la sintaxis concreta y asegurarse de que es válida y en segundo lugar usarla para construir el modelo de la sintaxis abstracta. Estas etapas son iguales tanto para sintaxis textual como gráfica, pero hay una diferencia importante en el modo en que se construye la sintaxis concreta. Los diagramas se construyen de manera interactiva e incremental, con lo que la comprobación se realiza en paralelo a dicha interacción, mientras que en la textual es un proceso por lotes una vez escrito el código. Por lo tanto deben definirse de manera precisa las correspondencias entre la notación de la sintaxis concreta y los elementos del metamodelo de la sintaxis abstracta.

En el caso de sintaxis textual se utilizan herramientas de tipo BNF (*Backus-Naur formalism*), mientras que en la gráfica los diagramas se interpretan incrementalmente, lo que supone un reto a la hora de escoger una arquitectura para soportarla. Para realizar esta interpretación primero hay que definir qué es un diagrama con cierto nivel de abstracción. Para ello hay modelos de diagrama como XMI de OMG para el intercambio de diagramas. El diagrama permite capturar las características de los conceptos sintácticos y sus relaciones. Un enfoque habitual consiste en usar herramientas, como EMF de Eclipse, que permiten traducir los conceptos y responder a la interacción con el usuario.

Cuando el usuario construye de manera interactiva el diagrama utilizando la sintaxis concreta, debe construirse el modelo de sintaxis abstracta de manera concurrente. En la práctica es deseable tener un mismo tipo de diagrama que genere diferentes sintaxis abstractas en diferentes circunstancias. Lo más flexible es tener correspondencias entre

la sintaxis concreta y la abstracta (herramientas como Eclipse mantienen esta sincronización, de manera que cuando se modifica el diagrama se modifica el modelo de sintaxis abstracta). Esta correspondencia se puede aplicar en ambos sentidos (también de abstracta a concreta) de manera que se mantengan actualizadas. De este modo podemos tener múltiples sintaxis concretas, o vistas, de manera que el cambio en una se actualice en las demás a través del modelo o sintaxis abstracta.

4.2.4 Semántica

La semántica de un lenguaje describe el *significado* de los conceptos que maneja, y permite comprender cómo usarlo. Existen diferentes formas de describir el significado de un concepto del lenguaje (considérese cómo se hace en los lenguajes naturales):

- En términos de conceptos que ya tienen un significado bien definido (un coche consiste de un chasis, ruedas, etc.).
- Describiendo las propiedades y comportamiento del concepto: "un coche puede estar aparcado o moviéndose, presionando el acelerador incrementa su velocidad".
- Como especialización de otro concepto: "un camión es un vehículo con trailer".
- Describiendo las propiedades comúnmente compartidas por todas las instancias de un concepto.

En lenguajes naturales, la semántica es una correlación o correspondencia entre conceptos del lenguaje con conocimiento o experiencias de conceptos del mundo que nos rodea. Aunque para programación es necesario un enfoque más formal, el paralelismo con los lenguajes naturales es perfectamente válido.

La semántica es un elemento clave para definir capacidades que son semánticamente ricas, como la ejecución, análisis y transformaciones, que son necesarias para soportar el desarrollo dirigido por modelos.

Tradicionalmente, la semántica se ha descrito de manera informal, mediante descripciones en lenguaje natural o ejemplos (como UML1.X). Sin embargo, la **semántica informal** tiene algunos problemas:

- Los usuarios deben asignar significado informal o intuitivo a los modelos, con el consiguiente riesgo de malas interpretaciones o un uso erróneo del lenguaje.
- No puede ser interpretado ni entendido por herramientas automatizadas. Los desarrolladores de herramientas deben aportar su propia interpretación lo que puede dar lugar a diferentes implementaciones de un mismo lenguaje.
- Hace difícil la tarea de definir nuevos lenguajes.
- Los estándares requieren semántica precisa.

Existen diferentes alternativas para realizar una descripción formal de la semántica de un lenguaje. Un enfoque muy frecuente es utilizar un lenguaje matemático formal. El inconveniente es que resulta difícil de comprender y su uso práctico está limitado. También es posible emplear un lenguaje de programación externo, pero compromete la independencia de la plataforma y obliga a salir del entorno de metamodelado para

usar un lenguaje, lo que hace el proceso de definición del lenguaje muy poco intuitivo. La solución más elegante consiste en **utilizar metamodelos**, que conlleva una serie de ventajas:

- La semántica está totalmente integrada con la definición del lenguaje lo que simplifica la relación con otros artefactos como la sintaxis abstracta o concreta y las relaciones.
- Como se usa el mismo lenguaje de metamodelado en todos los lenguajes, las definiciones semánticas se convierten en conjuntos reutilizables que se pueden integrar y extender con relativa facilidad.
- Las definiciones semánticas son **independientes de la plataforma de implementación**.

El metamodelo de la semántica es muy diferente del modelo de sintaxis abstracta del lenguaje, que define su estructura. Sin embargo el modelo de sintaxis abstracta es prerrequisito para definir la semántica, y la semántica añade una capa de significado a los conceptos definidos en la sintaxis abstracta.

La semántica debe distinguirse de la semántica estática, que son las reglas que determinan la corrección de las expresiones del lenguaje, y emplean, por ejemplo, los comprobadores sintácticos.

Existen numerosos enfoques para plantear la definición de la semántica. A continuación se presentan los cuatro más empleados en dominios de lenguajes de programación, que son trasladables a DSLs utilizando metamodelos para expresar las definiciones semánticas:

Traslacional. Consiste en trasladar conceptos de un lenguaje a conceptos de otro lenguaje que tiene una semántica precisa.

Operacional. Modela el comportamiento operacional de los conceptos del lenguaje.

Extensional. Extiende la semántica de los conceptos existentes de un lenguaje.

Denotacional. Modela las relaciones con los conceptos semánticos del dominio.

Cada uno tiene sus ventajas e inconvenientes. En la práctica se pueden utilizar varios enfoques combinados basándose en la naturaleza de los conceptos individuales del lenguaje. La semántica **Traslacional** se basa en dos nociones: la semántica se define cuando el lenguaje se traslada a otra forma (lenguaje destino), y el lenguaje destino se puede definir mediante un número reducido de constructores primitivos que tienen una semántica bien definida. El propósito es definir el significado del lenguaje en términos de conceptos primitivos que ya tienen su propia semántica bien definida, y que típicamente tienen una semántica operacional.

La principal ventaja es que es posible obtener directamente semántica ejecutable para el lenguaje a través de la traslación. El problema es que se puede perder información en

el proceso de transformación: la colección de primitivas resultante no se puede relacionar de manera obvia con los conceptos del modelo original de lenguaje. Esto se puede evitar mediante etiquetado en el lenguaje destino o bien manteniendo información de las relaciones entre modelos.

Existen distintas alternativas para incorporarlo a la definición del lenguaje:

- Dentro de un metamodelo del lenguaje trasladando un concepto a otro concepto que tiene una semántica definida donde las sentencias tienen una semántica operacional (por ejemplo, un *case* en una secuencia de *if* anidados).
- Entre metamodelos de lenguajes, transformando un metamodelo en otro. Por ejemplo, la sintaxis abstracta de UML se puede relacionar con un metamodelo de un lenguaje más pequeño y bien definido como XCore, o a un lenguaje de programación como Java.

La semántica **Operacional** describe cómo los modelos o programas escritos en un lenguaje pueden ser ejecutados directamente**.** Esto implica **construir un intérprete**. Al expresarse en términos de operaciones del propio lenguaje, al contrario de la traslacional (que se define en términos de otro lenguaje, posiblemente muy diferente) la semántica operacional puede ser más fácil de entender y escribir.

El poder definir un intérprete como parte del metamodelo requiere que el metamodelo sea ejecutable. Si este es el caso, los conceptos pueden definir operaciones que capturan su comportamiento operacional. Típicamente, la definición de un intérprete para un lenguaje sigue un patrón en el que se asocian conceptos con descripciones operacionales. Por ejemplo, para una máquina de estados, en el intérprete se puede definir una operación *run* que almacena en qué estado se encuentra la máquina (en primer lugar se fijará como estado actual el estado inicial de la máquina de estados). A continuación se entraría en un bucle *while* (cuya condición sería que no se ha llegado al estado final) que haría lo siguiente:

1. Invocar la acción de entrada del estado actual, llamando la operación de activación que se haya definido para éste (hay una descripción operacional para la acción del estado).
2. Determinar las transiciones de salida para ese estado y evaluar los eventos asociados. Cada evento tendrá una guarda (o condición) y una acción asociada (descripción operacional para evaluar los eventos y guardas asociadas).
3. Si a causa de un evento se dispara alguna transición (su condición es verdadera) se invoca la acción asociada al evento y después se establece como estado actual el estado destino (*target*) de la transición.

La principal ventaja de construir un intérprete es que se puede capturar un comportamiento complejo en términos de definiciones operacionales, además de que los modelos se pueden validar a partir de su ejecución.

La semántica **Extensional** se define como una extensión de otro lenguaje existente. Los conceptos de modelado en el nuevo lenguaje heredan la semántica de los conceptos del original. Además, se puede extender la semántica añadiendo, por ejemplo, nuevas capacidades. La ventaja es que se pueden reutilizar conceptos semánticamente complejos con el mínimo esfuerzo.

Este enfoque es similar al uso de perfiles en UML. Un perfil es una colección de estereotipos, que se pueden ver como subclases de elementos de modelado UML. Sin embargo, realizando la extensión a nivel de metamodelo se consigue mejor expresividad para el usuario, que puede añadir arbitrariamente extensiones semánticas al nuevo concepto. Así, la herramienta puede usar esta información para permitir implementación rápida de nuevos lenguajes de modelado. Puede reconocer que se ha producido una extensión y usar símbolos de estereotipos para personalizar los símbolos del lenguaje de modelado original para soportar el nuevo lenguaje.

Finalmente, la semántica **Denotacional** tiene como propósito asociar objetos matemáticos como números, tuplas o funciones con cada concepto del lenguaje. Se dice que el concepto *denota* el objeto matemático, y el objeto se llama *denotación* del concepto. Los objetos asociados al concepto son el *dominio semántico* del concepto. Se trata de una semántica por ejemplos, donde es posible definir el significado de un concepto proporcionando todos los posibles ejemplos. Algunos conceptos se denotan por una colección de ejemplos (*Integer* se denota por 0..infinito). Estas denotaciones tienden a ser estáticas y no ejecutables.

La semántica denotacional se puede definir en el metamodelo construyendo un modelo de la sintaxis abstracta del lenguaje y del dominio semántico, así como de la relación semántica entre ambos modelos. Entonces se escriben restricciones para describir cuándo las instancias de los conceptos del dominio semántico son válidos respecto de su sintaxis abstracta. En la práctica se usa cuando se requieren especificaciones semánticas de muy alto nivel. Muy útil en estándares, donde se quiere evitar un compromiso con una implementación específica (tal es el caso de la especificación de OCL 2.0).

En el ejemplo popular de la máquina de estados el modelo semántico podría incluir conceptos que describen su comportamiento, como cambio de estado (causado por una transición) y ejemplos o instancias de la máquina en estados específicos.

La elección de la semántica depende del tipo de lenguaje que se está definiendo. A continuación se proponen algunas directrices [Clark 08]:

- Si se dispone de una semántica declarativa y no ejecutable usar enfoque denotacional.
- Si el lenguaje tiene conceptos que deben ser evaluados, ejecutados o instanciados, deberían ser modelados usando un enfoque operacional, traslacional o extensional:

- Si el concepto es claramente una forma enriquecida de conceptos más primitivos, adoptar el enfoque traslacional. Esto evita tener que construir una semántica para el concepto desde la nada, reutilizando la semántica que ya ha sido definida para los conceptos primitivos. Este enfoque debe utilizarse sólo si es aceptable perder información sobre el concepto original.
- Si se debe mantener información sobre el concepto y es posible reutilizar un concepto existente, utilizar el enfoque extensional.
- Si no hay un medio adecuado para reutilizar un concepto existente, utilizar el enfoque operacional para construir un intérprete.

Como se ha visto, la semántica es esencial para poder entender el significado de un lenguaje de modelado, parar poder interactuar con él de manera significativa y para poder manejar de manera efectiva las definiciones del lenguaje con respecto de otros lenguajes destino. La semántica de un lenguaje puede capturarse de manera satisfactoria en un metamodelo. De esta forma, la semántica queda integrada como parte de dicho lenguaje, a la vez que es entendido por usuarios y por las herramientas con las que se integre.

En los siguientes apartados se presenta el DSL desarrollado para aplicaciones domóticas junto con el metamodelo que le da soporte y las herramientas empleadas para su diseño.

4.3 DSLs Existentes en Domótica

Uno de los motivos de la realización de este trabajo de Tesis es la práctica inexistencia de lenguajes específicos en el campo de la domótica que permitan una captura de requisitos con cierto nivel de abstracción e independencia de la plataforma. Tras una revisión en profundidad de la literatura y herramientas existentes, podemos distinguir entre dos tipos de propuestas: las independientes de plataforma (basadas en MDA), y las soluciones comerciales dependientes de plataforma. Entre las primeras, son de destacar los trabajos, anteriormente citados, de M. Voelter [Voelter 07] y J. Muñoz [Muñoz 06][Muñoz 07] dentro del ámbito de aplicación del enfoque de desarrollo por modelos.

De entre las numerosas herramientas comerciales (prácticamente una por cada tecnología domótica existente) se revisarán por su interés ETS (*Engineering Tool Software*) y LonMaker que son específicas de las plataformas KNX/EIB y Lonworks, respectivamente. Los trabajos de M. Voelter y J. Muñoz se comentaron en el apartado 3.6, pero no desde el punto de vista de lenguajes específicos de dominio.

4.3.1 DSLs Independientes de Plataforma Basados en MDA

M. Voelter propone una infraestructura para el diseño de sistemas domóticos que cubre todo el ciclo de vida, desde la captura de requisitos del sistema, pasando por transformaciones modelo a modelo, hasta la generación de código mediante transformaciones modelo a texto (M2T). El principal inconveniente de este trabajo radica en que **no se define un DSL**, sino que el experto del dominio debe utilizar un editor jerárquico (*Tree Editor*) de la herramienta Eclipse [Eclipse 08] para crear el modelo. Además, el metamodelo no contempla la adición de nuevos dispositivos domóticos, por lo que la inclusión de un nuevo tipo de dispositivo obligaría a la modificación del mismo. En la Figura 4-2 se muestra el metamodelo propuesto por M. Voelter y una captura del modelo en el formato *Tree Editor* (árbol desplegable).

En las publicaciones de J. Muñoz [Muñoz 06][Muñoz 07] se propone también la utilización de MDA para el ciclo de vida del diseño de sistemas domóticos. La captura de requisitos las realiza el analista del sistema mediante tres vistas: modelo de servicios, modelo de interacción y modelo estructural, a partir de los cuales un especialista de software genera la denominada vista de arquitecto. Para la creación de estas vistas se utiliza el lenguaje de modelado PervML. En la Figura 4-3 se presenta un ejemplo de dichas vistas, donde se puede apreciar cómo la captura de requisitos tampoco utiliza un DSL propiamente dicho, sino diagramas UML de clases (modelo de servicios), diagramas de interacción (modelo de interacción) y diagramas de componentes y servicios (modelo estructural). El principal inconveniente de utilizar diagramas UML en vez de un DSL es que son difíciles de entender por los especialistas del dominio domótico, que no lo son tanto en el campo de la Ingeniería del *Software*.

Por otra parte, sería deseable que el lenguaje permitiera la utilización de primitivas gráficas en su sintaxis concreta, de forma que facilitaría aún más la formación de los usuarios de la herramienta y la comprensión de los requisitos recogidos.

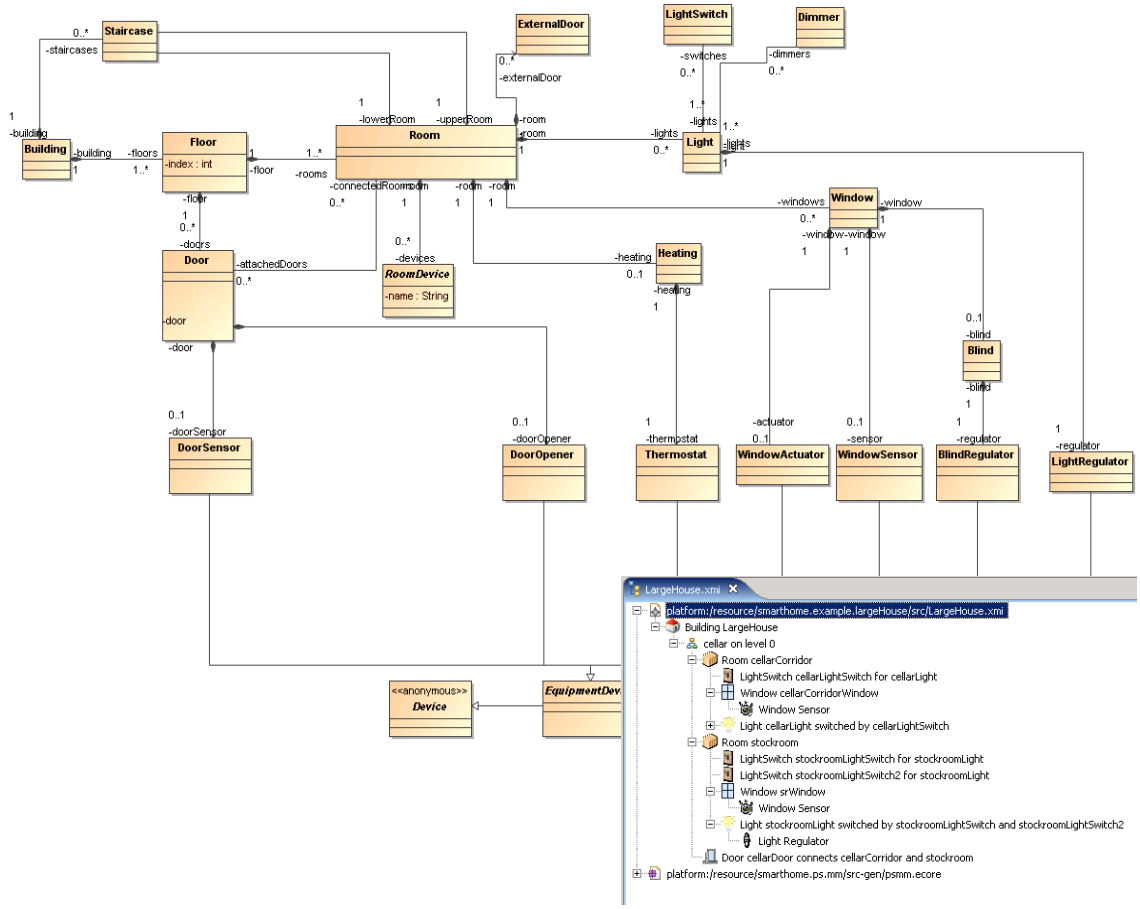


Figura 4-2. Metamodelo para aplicaciones domóticas y ejemplo de modelo con T*ree Editor* propuestos por M. Voelter. Extraído de [Voelter 07].

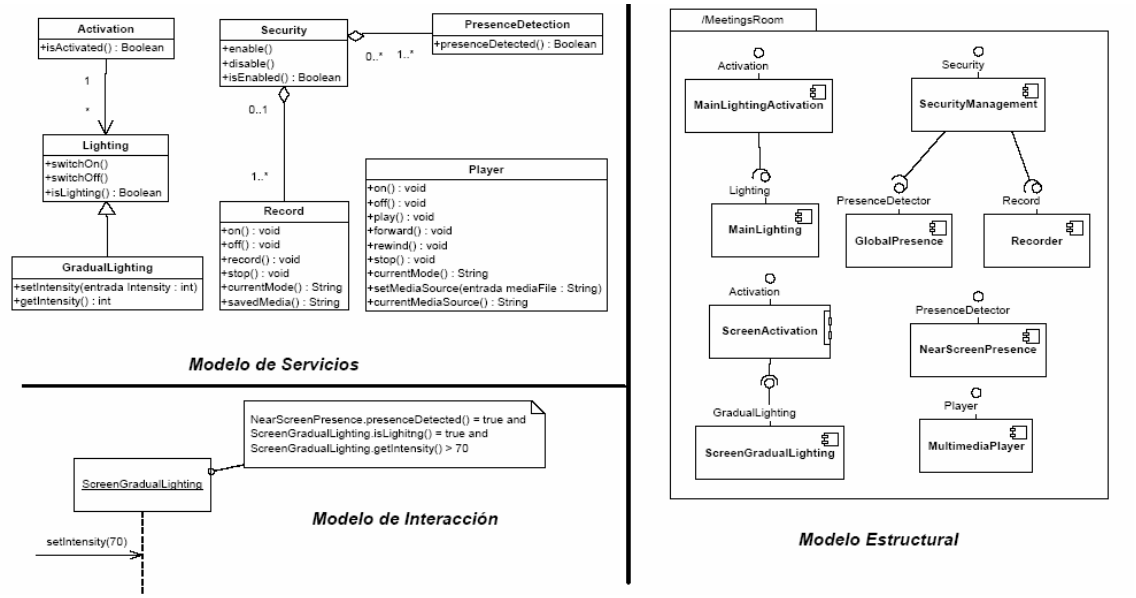


Figura 4-3. Modelo de servicios, interacción y estructural para sistemas domóticos propuesto por J. Muñoz. Fuente [Muñoz 06].

4.3.2 DSLs Específicos de Plataforma

En el ámbito de las herramientas específicas de plataformas comerciales son destacables las utilizadas para los dos estándares domóticos con mayor peso internacional: KNX/EIB (Konnex, recogido en normas ISO/IEC 14543-3-X) y Lonworks (recogido en normas EN 14908).

En el caso del sistema KNX/EIB, los desarrolladores disponen de una herramienta denominada ETS (*Engineering Tool Software*), de la que se muestra en la Figura 4-4 una instantánea de utilización. Se trata de un entorno de configuración, parametrización, programación y diagnóstico de dispositivos de esta plataforma basada en tres vistas relacionadas, que se mantienen sincronizadas:

- Vista de topología (ventana inferior derecha). En ella se especifica la topología de red de la instalación y los dispositivos incluidos en cada uno de los segmentos de red.
- Vista de edificios (ventana superior). Muestra los aparatos distribuidos por estancias (edificios, plantas, habitaciones, etc.).
- Vista de direcciones de grupo (ventana inferior izquierda). Detalla las direcciones de grupo creadas para realizar asociaciones entre dispositivos de la instalación.

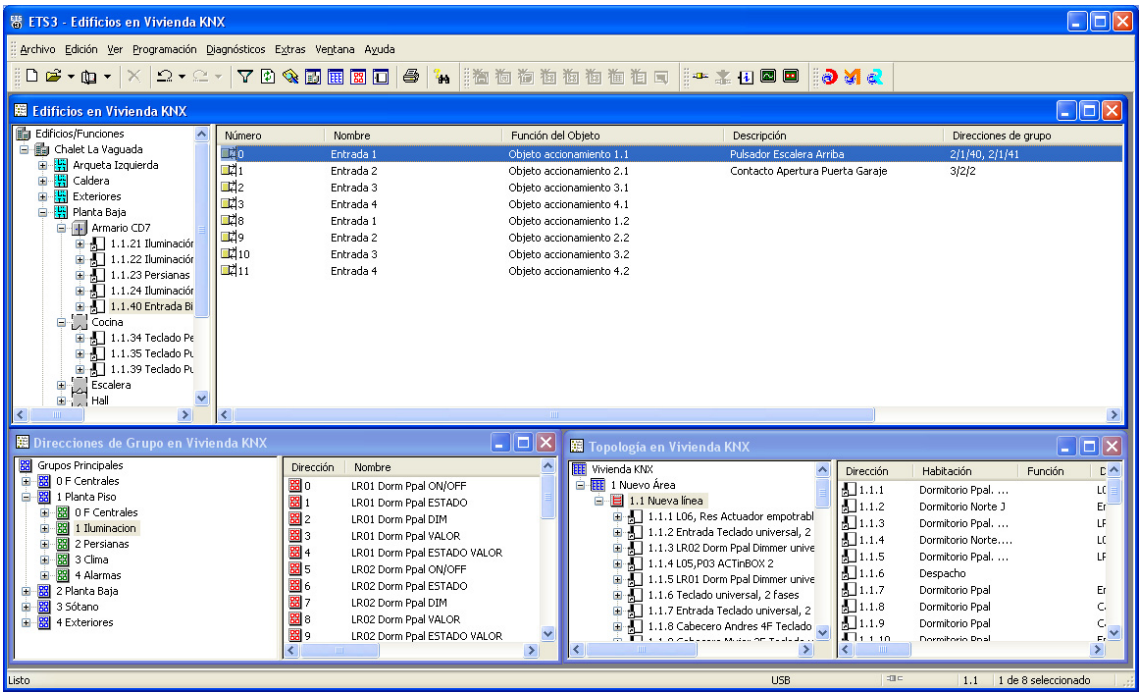


Figura 4-4. Captura de la herramienta *Engineering Tool Software* (ETS) para la tecnología KNX/EIB.

De esta manera, para el diseño de instalaciones KNX/EIB, el desarrollador debe crear una estructura de topología, de edificios y de direcciones de grupo. A partir de ella insertará aparatos domóticos de unos catálogos proporcionados por los diferentes

fabricantes de dispositivos de esta plataforma. Estos catálogos incluyen los *drivers* que se descargarán en los dispositivos y una interfaz para configurarlos. Así, cada aparato dispone de una funcionalidad predefinida (por ejemplo, un actuador binario permite encender y apagar elementos como bombillas o electroválvulas) que se parametrizará mediante la herramienta. Además, será necesario realizar enlaces lógicos entre los dispositivos a través de direcciones de grupo (que enlazan objetos de comunicación de los aparatos). Todos estos conceptos se describieron con detalle en el capítulo 2 "Sistemas Domóticos", apartado 2.5.5. "KNX/EIB".

Se puede concluir, por tanto, que se trata de un DSL **dependiente de plataforma**, **orientado a la generación de código** (enfoque c2 de la Figura 4-1), en el que se eleva poco el nivel de abstracción para la programación de los dispositivos. Además, la sintaxis concreta del lenguaje es muy poco intuitiva, y en muchos casos tiene que apoyarse en diagramas lógicos (no soportados) que representan los mismos conceptos que la herramienta, pero de manera gráfica, para comprender mejor las asociaciones que se realizan. Por otra parte, sólo se incluyen aquellos dispositivos que requieren programación, por lo que muchas veces los dispositivos finales (una bombilla, persiana o pulsador) no están reflejados. Esto supone una pérdida importante de información, que debe ser subsanada con la utilización de otros documentos, como planos o esquemas auxiliares, por lo que la semántica necesaria para comprender el funcionamiento del sistema se distribuye entre la información que recoge la herramienta (muchas veces de manera informal mediante comentarios) y otros documentos que no están integrados. Por ello, para la programación en esta plataforma es esencial una **formación muy especializada** y también es aconsejable cierta experiencia. Además, **no se permite la simulación**, por lo que la verificación del funcionamiento del sistema ha de realizarse sobre la instalación real.

La tecnología *Lonworks* utiliza la herramienta *LonMaker*, que se integra con *Microsoft Visio*, como se puede observar en la Figura 4-5. En este caso, la sintaxis concreta para el diseño del sistema es de tipo gráfico, gracias a la galería de símbolos disponibles para ello. Por lo tanto, se podría decir que se eleva un poco el nivel de abstracción respecto al sistema KNX/EIB, aunque no deja de ser un enfoque **orientado a la generación de código** (enfoque c2 de la Figura 4-1) que si bien mejora la interfaz con el usuario, adolece de algunos de los problemas descritos para la herramienta ETS, como la ausencia de los dispositivos finales y sobre todo la dependencia de plataforma y el **alto nivel de formación** requerido para el desarrollador. Además, **tampoco es posible realizar simulaciones** y la verificación del funcionamiento debe realizarse sobre la instalación real.

De la revisión de DSLs existentes para domótica se puede concluir que **las propuestas existentes basadas en MDA no ofrecen un DSL propiamente dicho**, y **las herramientas comerciales no proporcionan el suficiente nivel de abstracción** para modelar el sistema a alto nivel que permita obtener una implementación en distintas plataformas de manera automática. Por ello, se propone a continuación un DSL que solucione estos problemas.

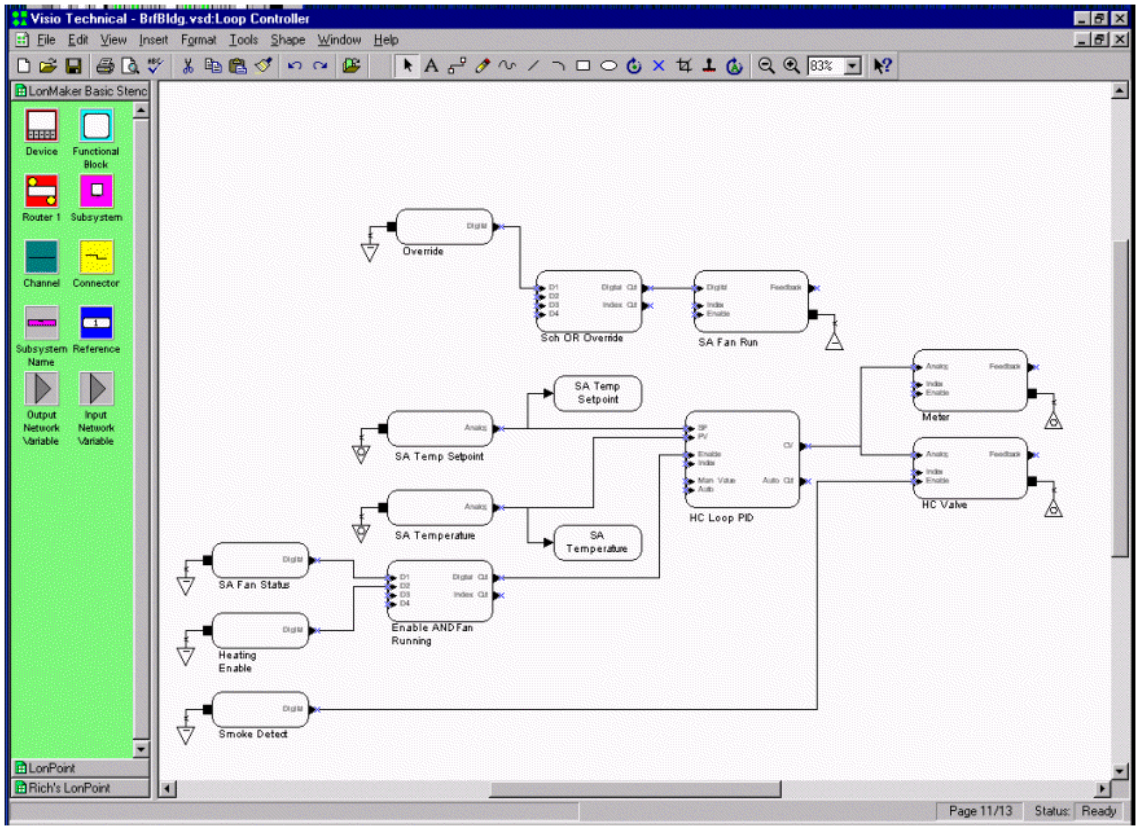


Figura 4-5. Captura de la herramienta *LonMaker* para la tecnología Lonworks.

4.4 Un DSL para Domótica

En este apartado se presenta un Lenguaje de Específico de Dominio que permite capturar los requisitos propios de un sistema de forma gráfica e intuitiva, utilizando primitivas propias del dominio domótico. Con ello se pretende mejorar el proceso de diseño actual en este ámbito (que se describirá en el capítulo 5), introduciendo las ventajas que aporta la utilización de DSLs, de entre las que cabría destacar:

- Empleo una notación gráfica cercana a la forma natural de pensar de los expertos del dominio.
- Independencia de la plataforma final de implementación, de modo que las soluciones definidas con el DSL puedan ser traducidas a diferentes estándares o tecnologías domóticas manteniendo la esencia de la definición de la aplicación. Con esto se evita que el desarrollador tenga que ser experto en los distintos estándares domóticos existentes.
- Generación automática o semiautomática del código de implementación para la plataforma seleccionada, con el aumento de productividad asociado.

4.4.1 Identificación de Conceptos

Tal como describen J. P. Tolvanen [Tolvanen 06] y T. Clark [Clark 08] la primera fase en la definición de un DSL es la identificación de abstracciones y relaciones entre ellas (sintaxis abstracta). Un sistema domótico se compone de elementos de los siguientes **tipos de dispositivos**: sensores, actuadores y controladores (véase el apartado 2.3.1):

- Los **sensores** son **dispositivos de entrada** que captan cambios en el entorno y los envían a los nodos de control, donde se toman las decisiones oportunas. Algunos ejemplos son: pulsadores, sensores de humedad, temperatura, luminosidad, detectores de humo, inundación, movimiento, etc.
- Los **controladores** son los nodos que procesan la información, coordinan y comunican las acciones a otros controladores o dispositivos.
- Los **actuadores** son **dispositivos de salida** que reciben las órdenes de los controladores y realizan las acciones sobre los dispositivos finales (pasivos): encender/apagar luces, subir/bajar persianas, abrir/cerrar electroválvulas, etc.

Estos elementos **aparecen en todas las tecnologías y estándares domóticos**, que se diferencian en la arquitectura, protocolos utilizados o módulos disponibles.

Dada la amplia variedad de dispositivos existentes, en el DSL propuesto se utiliza un catálogo de componentes domóticos reutilizables con la posibilidad de modelar nuevos componentes para ser incluidos en dicho catálogo. De esta manera se puede partir de los objetivos y procesos estratégicos del negocio, en vez de comenzar analizando el software desde una perspectiva tecnológica como ocurriría en niveles de abstracción inferiores.

Los conceptos fundamentales definidos en el DSL son los siguientes:

- **Unidades Funcionales**: son los elementos principales del sistema. Una unidad funcional es el elemento básico que puede formar parte de un dispositivo domótico, ya que dependiendo del fabricante, un dispositivo comercial puede integrar más o menos funcionalidad. La utilización del concepto de Unidad Funcional evita añadir dependencias con plataformas específicas. Por ejemplo, una unidad funcional puede ser un pulsador, una bombilla, una entrada binaria, una puerta lógica o un temporizador. Un dispositivo de una tecnología específica suele integrar una o varias de estas funciones.

  En un sistema domótico existen varios tipos de dispositivos, algunos de ellos restringen sus actividades al campo puramente hardware (bombillas, teclas de interruptores, etc.) y otros aportan la capacidad de implementación de servicios, así como la integración de los anteriores en el sistema. En este trabajo, aquellas unidades funcionales cuya actividad se reduce a su presencia física serán identificadas como *unidades pasivas finales* (una bombilla, un pulsador, un motor, una electroválvula). Este tipo de unidades funcionales ofrecen servicios de manera implícita (una bombilla ofrece el servicio de encender/apagar), pero su implementación no requiere de código software.

Estos dispositivos se encontrarán ubicados físicamente en algún lugar de la instalación (edificio, planta, habitación), por lo que también será necesario identificar dicha localización. Dada la naturaleza de las unidades funcionales, los de tipo final siempre tendrán una localización física en la instalación, mientras que los controladores, como elementos básicos que pueden formar parte de los dispositivos de la plataforma final de implementación, no tienen una localización definida a priori.

- **Servicios:** cada Unidad Funcional dispondrá de uno o varios servicios que serán utilizados para interaccionar con el resto de unidades funcionales. Siguiendo un símil a los servicios ofrecidos por una interfaz en otros dominios software, un servicio (*Service*) puede ser requerido o implementado. Por ejemplo, una bombilla deberá ofrecer, al menos, un servicio de conmutación (para encender/apagar) y opcionalmente uno o varios argumentos del servicio. Puesto que los servicios se utilizarán para enlazar diferentes unidades funcionales de la instalación, debe garantizarse la compatibilidad entre los mismos. Para ello se define también un **catálogo de servicios**, donde se establecen los tipos de servicios que tendrán que utilizar todas las unidades funcionales.
- **Parámetros**: las unidades funcionales deberán disponer de parámetros de configuración que permitan establecer cómo han de comportarse. Por ejemplo, una entrada binaria debe permitir conectar al sistema diferentes tipos de sensores. Un parámetro de configuración podría ser el comportamiento ante la generación de un flanco ascendente a su entrada, en cuyo caso podría realizar la llamada a un servicio de conmutación enviando un argumento de encendido, apagado o bien de cambio de estado respecto de su estado anterior (véase el ejemplo de la Tabla 4-3).
- **Catálogo de Unidades Funcionales.** Puesto que hay una serie de elementos que aparecen en todas las instalaciones domóticas, y con el fin de promover la reutilización de éstos, se ha creado un catálogo de Unidades Funcionales.

  El catálogo puede ir actualizándose gracias a la posibilidad de incorporar nuevos elementos, por lo que nunca quedará obsoleto. Este catálogo se compone de grupos de categorías que a su vez se organizan en subcategorías como se muestra en la Figura 4-6. Las unidades funcionales que se muestran son genéricas para cualquier instalación domótica (aunque no tienen por qué aparecer todas). Se distinguen dos grandes categorías: unidades funcionales finales (que son pasivas) y controladores (su implementación requiere código software):

  - Las **unidades pasivas** se clasifican en unidades de entrada y de salida. En las de entrada se incluyen todos aquellos sensores habituales en una instalación domótica: pulsadores, interruptores, detectores de movimiento, detectores técnicos (inundación, fuego, humo,…), así como aquellos que recogen variables ambientales como temperatura, humedad o nivel de luz. Las unidades de salida pasivas permiten modelar los aparatos sobre los que actúa el sistema, como bombillas de diversos tipos (regulables o no, incandescentes, fluorescentes), motorizaciones de persianas, toldos o pantallas, ventiladores

para climatización o electroválvulas para suministro de agua, gas o fluidos refrigerantes.

- La segunda categoría es la de los **controladores**, es decir, elementos que requieren programación para implementar su comportamiento. Los controladores se clasifican en cuatro subcategorías: entradas, salidas, entrada/salida y pasarelas. Los controladores de entrada permiten integrar dispositivos pasivos de entrada en el sistema domótico, para ello se dispone de entradas de conmutación y regulación, entradas para motorizaciones de persianas o entradas analógicas. Estas unidades deberán ofrecer (implementar) un servicio para conectar el dispositivo pasivo de entrada y requerir uno o varios servicios para su conexión a otras unidades funcionales. Por ejemplo, un controlador de entrada binaria se podrá conectar a un pulsador y podrá requerir el servicio de un actuador que esté conectado a una bombilla. Los controladores de salida realizan la misma función para los dispositivos pasivos de salida. Para ello se han creado las unidades de salida de conmutación, salida de regulación, salida para motorizaciones (*Shutter-Blind Output*) y salida analógica. Existen también controladores de entrada-salida, como puertas lógicas, temporizadores y filtros. Finalmente, las pasarelas (*Gateways*) permiten la comunicación con otras redes externas al sistema domótico (equipos infrarrojos, interfaces específicos como RS232, redes de telefonía para llamadas o envío de SMS, etc.).

En la Tabla 4-3 se muestra un ejemplo de definición de la unidad funcional *Switching In* y los servicios que requiere/implementa. Dicha unidad funcional se encuentra recogida en el catálogo de la Figura 4-7 dentro de la categoría *Controllers* y la subcategoría *Input*. La unidad funcional proporciona dos servicios. El primero de ellos (*SWIactivated*) se utiliza para detectar la activación de la unidad funcional pasiva que se enlace a este servicio, que pasará un argumento de tipo *booleano*. El comportamiento dependerá de los valores establecidos en los parámetros *SWIsendOnRisingEdge* y *SWIsendOnFallingEdge*. Éstos determinan con qué valor se realiza una llamada al servicio *SWIswitchOut* al detectar un flanco ascendente (*false* a *true*) o descendente (*true* a *false*) en el valor del argumento *OnOff* del servicio *SWIactivated*. El servicio provisto *SWIsetStatus* se utiliza para forzar un valor en el estado interno de la unidad funcional.

FUnit			
Name	*Switching In*	Ref	SWI
Provided Services	*SWIactivated(onOff: bool) : switch* *SWIsetStatus(OnOff: bool): switch*	*Detects input edge* *Sets the controller state*	
Required Services	*SWIswitchOut(onOff: bool) : switch*	*Calls an external switching service*	
Parameters	*SWIsendOnRisingEdge: { -|ON|OFF|Toggle}* *SWIsendOnFallingEdge: { -|ON|OFF|Toggle}*	*Value to send at rising edge* *Value to send at falling edge*	
Final	*False*		
Description	*Functional Unit to interface pushbuttons, switchs a binary detectors.*		

Tabla 4-3. Definición de la unidad funcional *Switching In*.

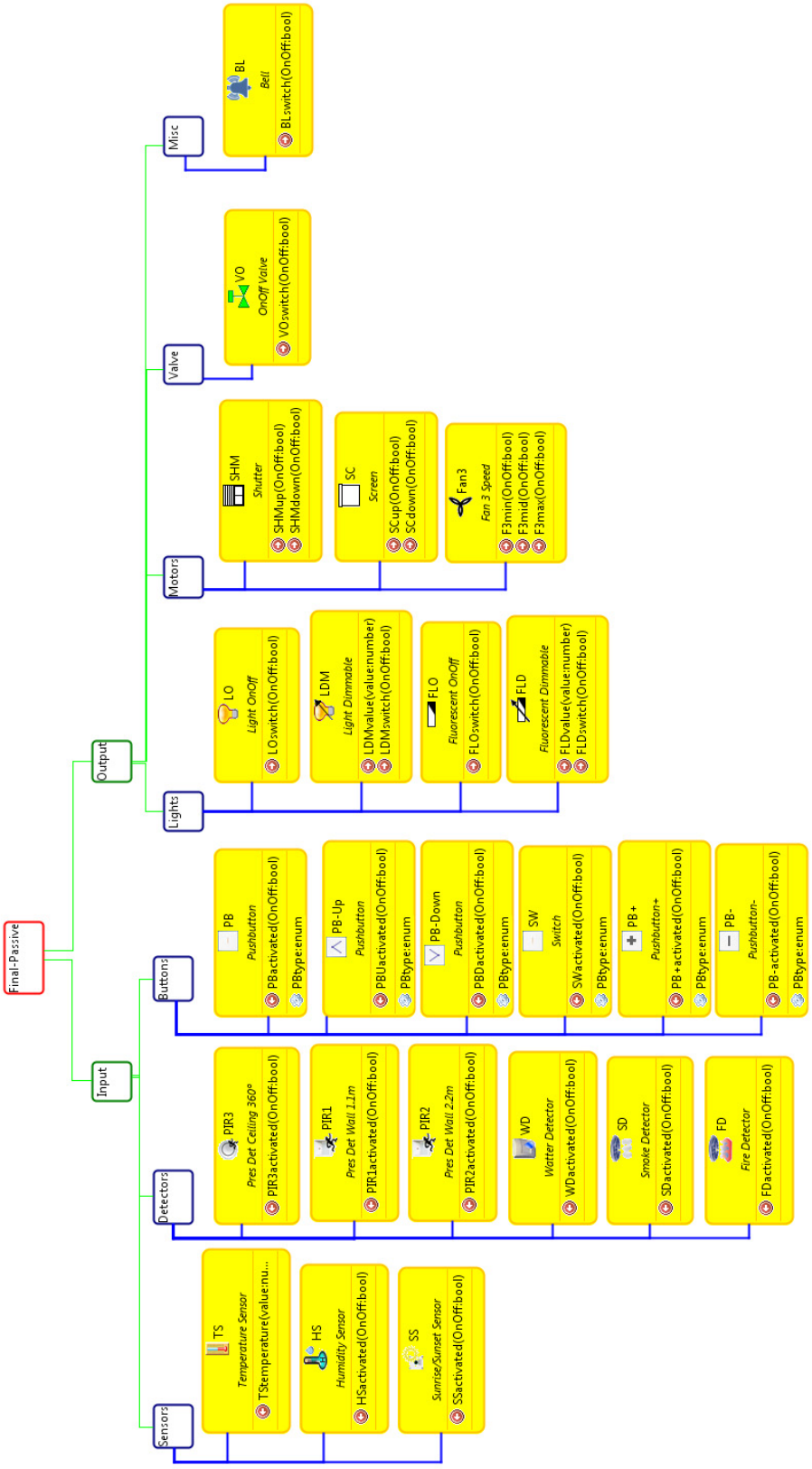


Figura 4-6. Proyección del catálogo de Unidades Funcionales: elementos pasivos.

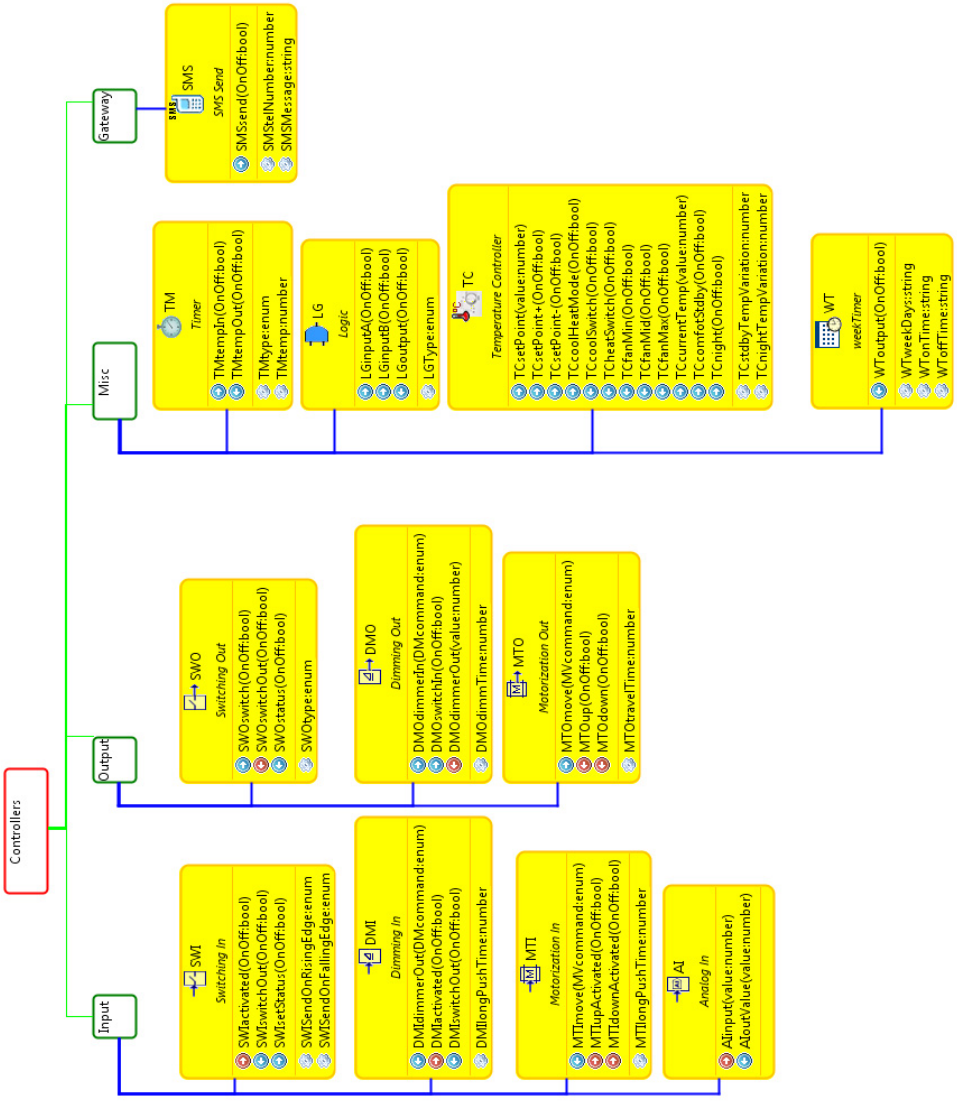


Figura 4-7. Proyección del catálogo de Unidades Funcionales: controladores.

- **Enlaces o asociaciones**. La manera habitual de relacionar dispositivos en una aplicación es utilizar enlaces. Así, mediante estas asociaciones se puede describir cualquier acción-reacción de la instalación domótica basada en la generación de eventos en el entorno. En la Figura 4-8 se muestra un ejemplo sencillo de asociación para el encendido de una bombilla (LO-1) como reacción a la activación de un pulsador (PB-1) por parte del usuario del sistema. Para ello sería necesario insertar la unidad funcional pulsador, que se conecta a una instancia de la entrada de conmutación SWI-1. Esta conexión se ha representado en rojo por tratarse de una conexión *hardware* (cableada entre aparatos que serán hardware de la implementación final). A su vez, la unidad de entrada binaria tiene asociado un comportamiento en su definición, de manera que cada vez que se activa el pulsador realiza una llamada a un servicio de conmutación, que se encuentra implementado en la unidad de actuación binaria (SWO-1), que activa la bombilla (LO-1). Para describir este ejemplo se han adelantado algunos elementos de la sintaxis concreta

del DSL, que se describirán más adelante. Así, sería posible realizar cualquier tipo de asociación entre unidades funcionales de la instalación domótica para llevar a cabo funciones mucho más complejas.

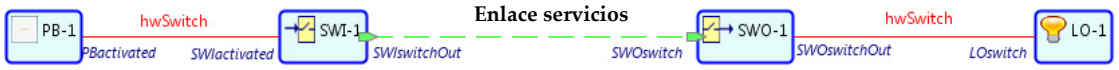


Figura 4-8. Enlaces entre unidades funcionales.

- **Escenas**: Las escenas son un conjunto de eventos encadenados que se generan como consecuencia de una única orden. Por ejemplo, se puede programar una escena "ir a dormir". Un único evento activará la escena y se irán ejecutando de forma secuencial una sucesión eventos o pasos de escena: apagar luces, bajar persianas, cerrar gas,…. En la Figura 4-9 se muestra un ejemplo de escena "presentación" con tres pasos: la escena se inicia mediante la llamada de un servicio de activación desde un pulsador. En el primer paso se regula la iluminación de la sala en el segundo se baja la veneciana y en el tercero la pantalla de proyección.

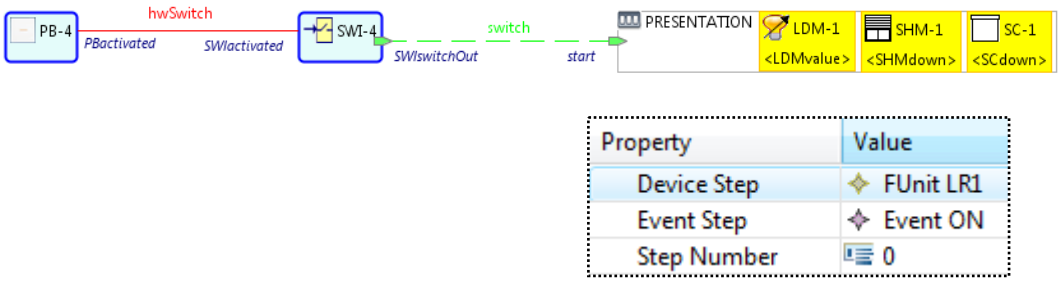


Figura 4-9. Ejemplo de escena.

Una vez identificados los conceptos más importantes que se van a utilizar en el dominio de la domótica, se procederá a describir la sintaxis abstracta y las reglas que permitan modelar aplicaciones empleando estos conceptos siguiendo un enfoque dirigido por modelos.

4.4.2 Sintaxis Abstracta. Metamodelo de Soporte al DSL

Todo Lenguaje Específico de Dominio debe tener una sintaxis abstracta que defina de manera formal los conceptos del lenguaje y las relaciones entre ellos. Para ello se ha partido de los conceptos identificados en el apartado anterior. En la Figura 4-10 se muestra una representación gráfica del metamodelo que define la sintaxis abstracta del lenguaje específico del dominio domótico (DSL) presentado anteriormente. Un metamodelo establece los conceptos y relaciones, incluyendo además las reglas que determinan cuándo un modelo está bien formado. En resumen, un metamodelo es un modelo del dominio.

Las herramientas que se han desarrollado para dar soporte a la creación de modelos hacen uso de este metamodelo como repositorio con el que tener almacenados los modelos, además de proporcionar los recursos necesarios para tareas de verificación y validación de los mismos. En este trabajo se ha utilizado el entorno EMF (*Eclipse Modeling Framework*).

Con el objetivo de obtener una mayor modularidad se ha optado por separar el metamodelo en dos partes, la parte utilizada para el catálogo de unidades funcionales (izquierda de la Figura 4-10) y una segunda parte para la aplicación domótica (derecha de la Figura 4-10). De este modo, el catálogo se modela de manera independiente a la aplicación, así se posibilita su reutilización. Ambas secciones (catálogo y aplicación) están relacionados a través de una serie de asociaciones destinadas a facilitar el uso (instanciación) de las clases del catálogo que definen conceptos reutilizables (instanciables). Por lo tanto, el metamodelo dará soporte a la sintaxis abstracta que permita construir tanto modelos del catálogo de unidades funcionales, como a modelos de aplicaciones. Para tal fin, se ha creado una herramienta para cada tipo de modelo. Los actores implicados en su uso serán:

- **Experto en el diseño de dispositivos**, quien diseñará el catálogo de unidades funcionales reutilizables para las aplicaciones. Podría tratarse bien de un experto que cree las definiciones genéricas de unidades funcionales a partir de las existentes en el mercado domótico, o bien un fabricante de dispositivos de alguna plataforma que proporcionara estas definiciones genéricas para sus dispositivos.
- **Diseñador de aplicaciones o experto del dominio**. Será el encargado de definir aplicaciones mediante instanciación, interconexión y configuración de las unidades funcionales proporcionadas en el catálogo. Gracias a la definición de un DSL gráfico (véase el apartado de sintaxis concreta) este usuario será conocedor del dominio domótico, pero no tiene por qué ser un experto en el mismo.

El metamodelo se basa en cinco tipos de elementos principales: unidades funcionales (*FUnitDefinition*, *FUnitInstance*), catálogo de unidades funcionales (*FUnitCatalog*), catálogo de servicios (*ServiceCatalog*), enlaces (*FULink*) y las escenas (*Scene*), que a continuación se describen con mayor detalle.

Unidades Funcionales

Como se explicó en el apartado anterior, las Unidades Funcionales son los elementos básicos del sistema (bombilla, persiana, etc.). En el metamodelo podemos encontrar dos elementos para modelarlas: *FUnitDefinition* y *FUnitInstance* que permiten modelar la definición y la instancia respectivamente de las unidades funcionales. La definición de unidad funcional (*FUnitDefinition*), tiene cinco atributos:

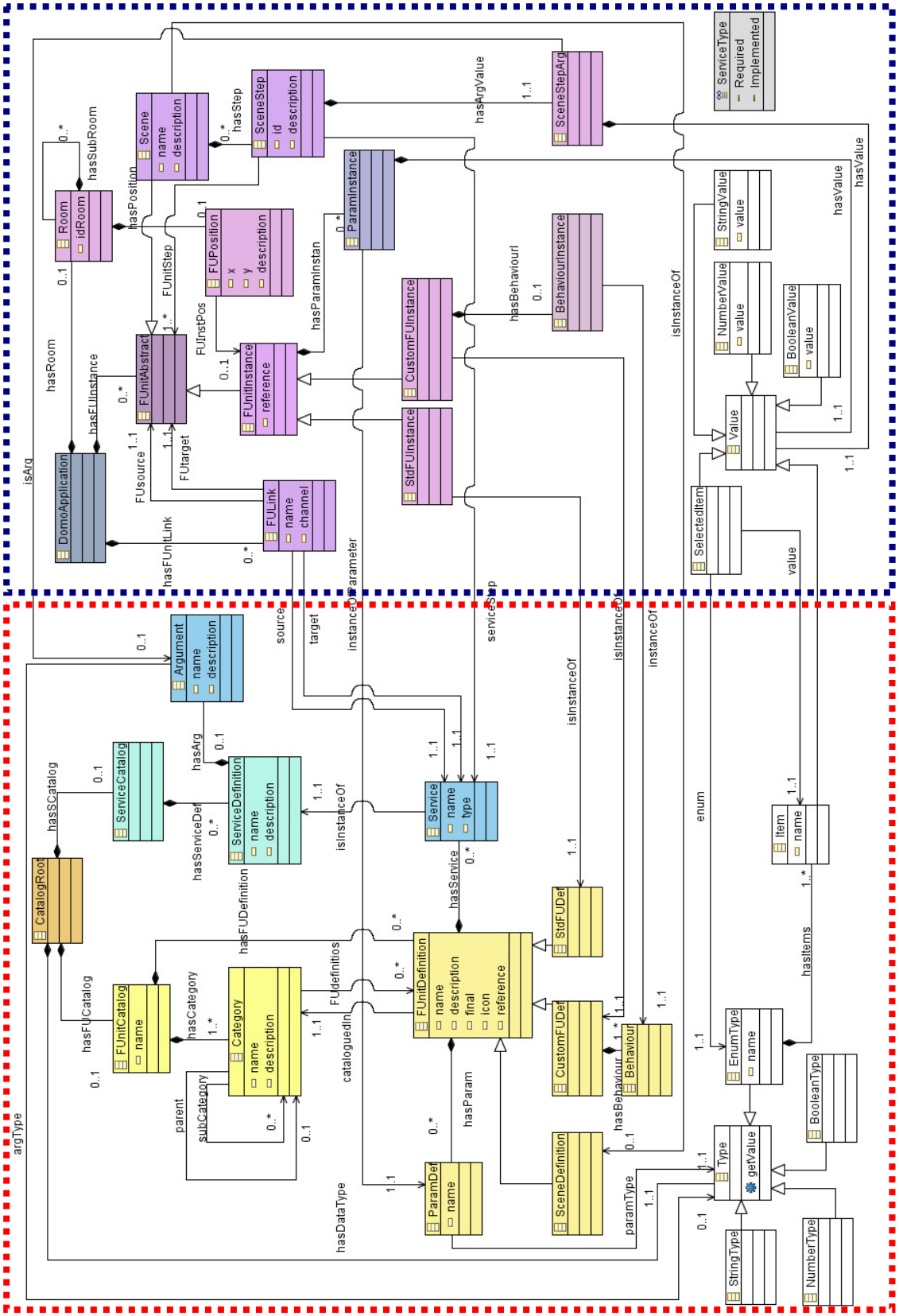


Figura 4-10. Metamodelo para el DSL.

- *name*: representa el nombre de la unidad funcional, es obligatorio y no puede estar repetido ya que se comporta como su identificador.
- *description*: permite describir (de forma opcional) las principales características de la unidad funcional.
- *final*: indica si la unidad funcional es una unidad pasiva-final o no.
- *icon*: es un atributo opcional donde se puede especificar el icono que se debe mostrar en el DSL. En el caso de no indicar ninguno se mostrará un icono por defecto.
- *reference*: permite introducir una referencia única para facilitar la posterior numeración de las instancias (por ejemplo PB-1, PB-2, etc.).

Además, esta definición contendrá una serie de **parámetros** (*ParamDef*) que pueden ser del tipo cadena de caracteres, numérico, booleano o enumerado y también contendrá los **servicios** (*Service*) tanto implementados como requeridos por la unidad funcional. En la Tabla 4-4 se muestra un ejemplo con la definición de una unidad funcional para un pulsador convencional. Esta unidad requiere de un servicio de conmutación (*PBactivated*) que es de tipo *switch* (véase el catálogo de servicios) y utiliza un argumento para indicar si está activo o no (*OnOff*). Además, dispone de un parámetro que indica si es de tipo normalmente abierto (*NO*) o normalmente cerrado (*NC*). La unidad es de tipo *Final*, por lo que se trata de un elemento pasivo (elemento *hardware* que no incluye código de programación) que se conectará a otras unidades activas mediante un canal de conexión.

FUnit			
Name	*PushButton*	Ref	PB
Provided Services	-		
Required Services	*PBactivated(onOff: bool) : switch*	*Switching service call*	
Parameters	*Type: { NO | NC }*		
Final	*True*		
Description	*Standard pushbutton*		

Tabla 4-4. Definición de la unidad funcional *Pushbutton*.

Por otro lado, para la instanciación de la unidad funcional se dispone de la clase *FUnitInstance* que mantiene una asociación con la definición (*FUnitDefinition*). De esta manera la instanciación mantiene toda la estructura e información que se ha representado en la definición con la ventaja de poder instanciar la misma unidad funcional múltiples veces. Al igual que ocurre con la definición, la instancia de la unidad funcional contendrá un ejemplar de los parámetros, que contiene a su vez los valores que se asignan a dichos parámetros al definir el modelo del sistema con el DSL.

Catálogo de unidades funcionales

El concepto de catálogo es uno de los más importantes ya que el principal objetivo es promover la reutilización de los elementos domóticos (unidades funcionales). Un catálogo de unidades funcionales (*FUnitCatalog*) se compone de una serie de categorías (*Category*) identificadas de forma unívoca por su nombre. Estas categorías pueden

agruparse de forma jerárquica permitiendo obtener el catálogo de la Figura 4-6. El catálogo también contiene las definiciones de las unidades funcionales (*FUnitDefinition*) y la definición de los tipos de datos disponibles para los parámetros de las unidades funcionales, mediante la clase *Type*, que se especializa en *StringType, NumberType, BooleanType y EnumType*. Los tipos enumerados (*EnumType*) contendrán una lista de literales (*Item*) de los cuáles se podrá seleccionar uno (mediante *SelectedItem*). Tanto *Item* como *SelectedItem* son especializaciones de la clase *Value*, por lo que representarán valores de los tipos especializados de dicha clase.

El metamodelo que da soporte a la sintaxis abstracta contempla la posibilidad de crear y modificar este catálogo sin que ello suponga modificar el metamodelo. Así, la utilización de las unidades funcionales se materializa mediante instancias a las mismas, lo que permite dar cierta estabilidad al metamodelo, y en consecuencia, a las herramientas desarrolladas.

El DSL propuesto ofrece la posibilidad de crear nuevos dispositivos domóticos. Por esta razón se especializan tanto la definición como la instanciación de una unidad funcional en dos elementos, el de tipo estándar y el personalizado (*custom*). Las unidades funcionales de nueva creación se modelarán haciendo uso del elemento *CustomFUDef* donde se tendrá que indicar el comportamiento (*Behaviour*) concreto de esa unidad funcional.

Catálogo de servicios

Las unidades funcionales disponen de servicios que representan la funcionalidad concreta con la que una unidad funcional trabajará. Una unidad funcional puede tener múltiples servicios que serán utilizados para interaccionar con el resto de unidades funcionales, por lo que los servicios que ofrecen o implementan las unidades funcionales deben ser compatibles entre sí. Por ello se ha definido un catálogo de servicios (*ServiceCatalog*) que recoge todas las definiciones de los servicios existentes (*ServiceDefinition*), de este modo se garantiza la compatibilidad de servicios entre unidades funcionales por un lado, y se recogen los servicios de uso habitual en el dominio de la domótica por otro. Los servicios se instanciarán tantas veces como sea necesario en la unidad funcional que lo necesite. De esta manera se consigue aumentar la reutilización y definir un formato estandarizado para establecer relaciones entre dispositivos, ya que un servicio requerido sólo se podrá conectar a uno implementado del mismo tipo.

La definición de servicio (*ServiceDefinition*) contiene dos atributos:

- *name*: representa el nombre de la definición de servicio, es obligatorio y no puede estar repetido ya que se comporta como el identificador.
- *description*: este atributo recoge la descripción donde se describe brevemente de forma opcional las principales características de la definición del servicio.

Una definición de servicio contendrá los argumentos (*Argument*) necesarios. Un ejemplo de definición de servicio puede ser: switch(*onOff: boolean*), donde switch es el nombre identificador del servicio y *onOff* el argumento del tipo booleano.

Cada servicio (*Service*) puede ser requerido o implementado. Esto se indicará en sus atributos que pueden ser:

- *name*: representa el nombre del servicio, es obligatorio y no puede estar repetido ya que se comporta como identificador.
- *type*: este atributo indica si el servicio es requerido o implementado. Puede tomar uno de los valores recogidos en el enumerado *ServiceType (Required o Implemented)*

Enlaces o asociaciones

Un enlace (*FULink*) permite interconectar las unidades funcionales instanciadas e indicar la forma en la que van a interactuar dichas unidades con el resto del sistema. Un enlace puede comportarse de dos maneras distintas dependiendo del tipo de unidades que esté enlazando. Se comportará como un **canal de conexión** cuando una de las unidades funcionales que enlaza sea pasiva-final, es decir, un *FULink* puede modelar una conexión hardware (por ejemplo, un enlace entre un actuador de iluminación y una bombilla). Este comportamiento se indica en el atributo *channel* de manera que si el valor de este atributo es *true*, el enlace se comportara como un canal de conexión. En el resto de casos se comportará como un enlace normal (por ejemplo, un enlace entre una entrada binaria y un actuador de iluminación).

El enlace asocia dos unidades funcionales, la unidad funcional origen (*source*) y la unidad funcional destino (*target*), y a su vez asocia dos servicios, pero no dos servicios cualquiera, un servicio origen que será un servicio requerido o implementado por la unidad funcional origen y un servicio destino que será implementado o requerido por la unidad funcional destino, siendo estos dos servicios del mismo tipo.

Escenas

Para la definición de escenas se utiliza la clase *Scene*, que es una especialización de *FUnitAbstract*, e instancia la definición de escena del catálogo (*SceneDefinition). SceneDefinition* se emplea para definir este tipo "especial" de unidad funcional que sólo tendrá un servicio provisto (de activación/conmutación). La escena contendrá uno o más pasos de escena (*SceneStep*), cada uno de los cuales realizará una llamada a un servicio específico de una instancia de unidad funcional del sistema, con un valor definido para su argumento (para ello se han definido la clase *SceneStepArg* que contiene el valor del argumento para la llamada al servicio y una referencia al argumento de que se trata).

4.4.2.1 Reglas OCL

Una parte fundamental en la construcción del metamodelo para la sintaxis abstracta es la definición de reglas que garanticen que las expresiones creadas a partir de él son correctas. Estas reglas permitirán verificar si un modelo satisface las restricciones definidas por su metamodelo, de manera que se pueda afirmar que el modelo es correcto con respecto a dichas reglas. Estas restricciones están asociadas a una determinada metaclase del modelo, concretamente a la que se ve afectada por la restricción.

Para especificar las restricciones se utiliza OCL [OCL 06]. Como el metamodelo es muy extenso, este trabajo se va a centrar en unas pocas restricciones para ilustrar brevemente la verificación de los modelos.

Restricciones para nombres

La mayoría de los elementos del metamodelo tienen un atributo *name* donde se almacenan los nombres de estos elementos que son utilizados como identificador único del elemento, por esta razón se debe imponer una restricción que obligue al usuario a preservar la identificación en todo momento de estos elementos, de manera que no se repita nunca un identificador.

```
self.Category->forAll(n1,n2| n1<>n2 implies
                        n1.name<>n2.name)
self.FUnitDefinition->forAll(fud1,fud2| fud1<> fud2 implies
                        fud1.name<>fud2.name)
self.ServiceDefinition->forAll(sd1,sd2| sd1<> sd2 implies
                        sd1.name<>sd2.name)
```

Como puede observarse, para los elementos del metamodelo *Category*, *FUnitDefinition* y *ServiceDefinition* se impone que no existan dos instancias con el mismo nombre.

Restricciones para enlaces

Otra restricción importante es la referente a los enlaces. En los enlaces los servicios que entran en juego deben complementarse entre ellos, de manera que se enlace un servicio implementado de una unidad funcional con un servicio requerido de otra unidad funcional o viceversa. No tiene sentido enlazar dos servicios del mismo tipo (requerido con requerido o implementado con implementado). Esto se realiza mediante la siguiente regla OCL:

```
self.FULink.->forAll(f | f.source.type!=f.target.type)
```

Esta regla recorre todos los enlaces entre unidades funcionales comprobando que el tipo de servicio en su origen (provisto o requerido) es diferente del tipo de servicio en su destino.

4.4.2.2 Conclusión

El metamodelo presentado define la sintaxis abstracta y las reglas para construir modelos de aplicaciones domóticas basándose en conceptos del dominio independientes de la plataforma. La separación del catálogo de la aplicación en sí permite la reutilización de las definiciones de unidades funcionales que son comunes en todos los sistemas domóticos. Para soportar la variabilidad, el diseñador de dispositivos puede definir nuevas unidades funcionales para su empleo en nuevas aplicaciones o ampliación de las ya existentes.

De entre los enfoques propuestos por J. Luoma [Luoma 04], el DSL presentado se encuadraría entre los basados en conceptos utilizados por los expertos del dominio (c1) y en espacio de variabilidad (c4), al apoyarse en la definición de un catálogo ampliable de componentes reutilizables. El catálogo de unidades funcionales recoge los elementos comunes y la variabilidad requerida para las diferentes aplicaciones se soporta mediante la implementación particular de cada sistema mediante interconexión y configuración de unidades funcionales, así como por la posibilidad de definir nuevas unidades funcionales (*CustomFUDef*). El empleo de conceptos del dominio facilita la formación de usuarios del lenguaje utilizando nociones presentes en todos los sistemas domóticos y, además, permite realizar implementaciones en distintas plataformas sin necesidad de modificar el DSL.

Entre las limitaciones cabe destacar que sería deseable disponer de una vista de localización (ubicación de dispositivos en habitaciones). El metamodelo recoge la posibilidad de ubicar los dispositivos finales en estancias. El problema es la ubicación de las unidades funcionales no finales que es dependiente de la tecnología y habrá de realizarse en la generación de código para las plataformas específicas. También sería conveniente que el usuario o el diseñador tuviera la posibilidad de definir unidades funcionales complejas a partir de diseños ya realizados (que incluyeran unidades funcionales básicas y enlaces entre ellos, convirtiendo el diseño en una "caja negra" reutilizable). Dada la complejidad de abordar este problema desde el punto de vista de la herramienta utilizada, esta característica se soporta, de momento, mediante la función "copiar-pegar" de partes del diseño entre diferentes modelos.

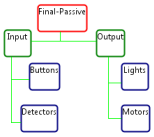
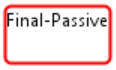
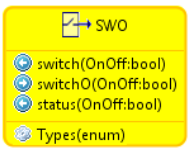
En el siguiente apartado se presentará una de las características que hacen más atractivo el DSL: la notación gráfica utilizada para representar los elementos descritos en esta sección.

4.4.3 Sintaxis Concreta

Una vez establecidos el vocabulario y la gramática del lenguaje mediante la sintaxis abstracta, ha de definirse cómo ser presenta al usuario. De entre los planteamientos posibles para la construcción de la sintaxis concreta se ha optado por la definición de una interfaz gráfica para el lenguaje con el fin de capturar, de la manera más intuitiva posible, los conceptos del sistema en el modelo y facilitar la formación de los usuarios. Por supuesto, no es posible expresar todos los conceptos de manera gráfica, al menos sin complicar excesivamente la interfaz, por lo que determinados artefactos se recogerán de manera textual como propiedades de los iconos gráficos (muchos de los atributos de las clases definidas en el metamodelo serán accesibles mediante la interacción con la clase, pero no visibles en el diagrama).

En la Tabla 4-5 se muestran los elementos gráficos utilizados para la representación de los conceptos utilizados en el DSL del catálogo. En la Tabla 4-6 se detallan algunos de los elementos gráficos empleados para la creación de aplicaciones. La definición de éstos se basa en la forma habitual que utilizan los expertos del domino para representarlos (muchas veces de manera informal para después traducirlos al lenguaje utilizado por las herramientas comerciales). Los ejemplos del apartado "4.4.1 Identificación de Conceptos" ya se expresaron utilizando este tipo de notación.

En el anexo A se muestra una aplicación en la que se utiliza esta sintaxis concreta de tipo gráfico para definir un sistema domótico completo y evaluar su idoneidad.

Icono	Representación Gráfica	Descripción
		**Catálogo unidades funcionales:** compartimento donde queda recogidas todas las categorías y definiciones de unidades funcionales.
		**Categoría**: especialización de un elemento del catálogo.
		**Enlace categoría-categoría:** permite enlazar las distintas categorías y así formar un catálogo jerárquico.
		**Definición de Unidad Funcional**: es el elemento más pequeño en que se puede subdividir un dispositivo domótico. Incluye un icono, un nombre y los servicios.

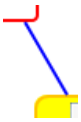
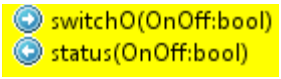
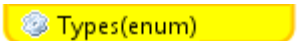
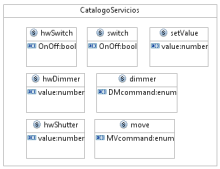
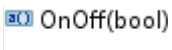
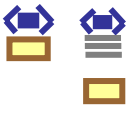
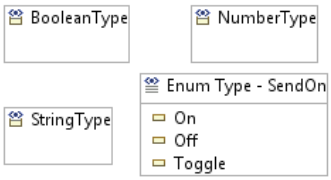
Icono	Representación Gráfica	Descripción
		**Enlace unidad funcional-categoría**: permite catalogar una unidad funcional dentro de una categoría del catálogo.
		**Servicio**: forma parte de una definición de unidad funcional. El servicio se muestra indicando si es provisto o requerido (fleha a la derecha o a la izquierda) y si es o no de tipo *hardware* (en color rojo o azul).
		**Definición de Parámetro**: para mostrar los parámetros de una definición de unidad funcional.
		**Escena**: es una especialización de unidad funcional. Se utiliza para modelar escenas.
		**Catálogo servicios**: compartimento donde quedan recogidas todas las definiciones de servicios.
		**Definición de servicio**: permite definir los servicios que utilizarán las unidades funcionales. Esta definición contiene los argumentos de entrada disponibles para el servicio.
		**Argumento**: se crea en una definición de servicio. Muestra entre paréntesis el tipo de dicho argumento.
		**Tipos**: permiten la creación de los tipos básicos que se reutilizan en los argumentos y los parámetros.

Tabla 4-5. Definición de la sintaxis concreta (gráfica) para el DSL del catálogo.

A continuación se describen las herramientas utilizadas para la construcción del metamodelo de sintaxis abstracta, la creación de reglas del lenguaje y el diseño del DSL gráfico acorde con dicho metamodelo.

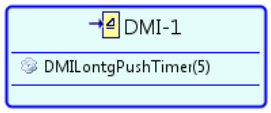
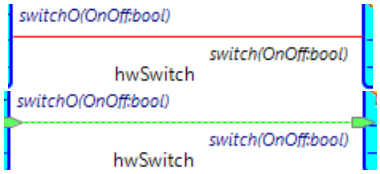
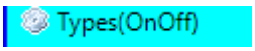
Icono	Representación Gráfica	Descripción
		**Instancia de unidad funcional**: muestra el icono de la unidad funcional que se instancia junto con los parámetros y sus valores.
		**Enlace entre unidades funcionales**: dependiendo de si se trata de un canal o un enlace, se representa como una línea roja que conecta dos unidades funcionales (una de ellas pasiva-final) o con una línea discontinua verde con terminaciones. En dicho enlace se muestran tres etiquetas, una para cada servicio que participa en el enlace y una tercera etiqueta central que muestra la definición del servicio que actúa en el enlace.
		**Escena**: contiene los pasos de escenas que se ejecutarán cuando arranque dicha escena.
		**Paso de escena**: muestra el servicio que se llamará junto con el incono y referencia de la unidad funcional a la que pertenece.
		**Parámetro**: se representan dentro de las unidades funcionales y a su lado el valor que se le ha asignado.

Tabla 4-6. Definición de la sintaxis concreta para el DSL de desarrollo de aplicaciones.

4.5 Soporte al DSL en un Marco de Gestión de Modelos

Debido a la gran variedad de artefactos software, la interoperabilidad con herramientas de modelado de cualquier índole constituye una condición necesaria para que una herramienta de gestión de modelos tenga éxito. MDA promueve el uso de una serie de estándares que permiten compartir meta-información entre herramientas de modelado (XMI [XMI 05]), especificar restricciones semánticas (lenguaje OCL [OCL 06]), especificar transformaciones de modelos (enfoque QVT [MOF-QVT 05]) y, sobre todo, proporciona un lenguaje abstracto y común (MOF [MOF 04]) con el que definir lenguajes de modelado. Más en concreto proporciona una sintaxis precisa con la que poder crear los metamodelos.

La herramienta que se ha desarrollado en este trabajo de Tesis para dar soporte a la metodología propuesta hace uso del entorno de desarrollo Eclipse [Eclipse 08] que ofrece un marco de trabajo donde poder crear meta-modelos y modelos. Eclipse incorpora múltiples proyectos relacionados con MDE, permitiendo llevar a cabo tareas de modelado, transformaciones de modelos, verificación, generación de entornos gráficos, generación de código, etc.

Como base tecnológica se utiliza la herramienta de modelado EMF [Steinberg 08] incorporada como plugin en Eclipse. EMF permite crear editores de modelos y

suministra las bases para la interoperabilidad con otras herramientas y aplicaciones basadas en MOF. Sobre la infraestructura ofrecida por EMF se ha utilizado la herramienta de generación de editores gráficos GMF (*Graphical Modeling Framework*) [GMF 08] que surge con el objetivo de generar automáticamente editores gráficos como *plugins* para Eclipse a partir de modelos.

GEF (*Graphical Editing Framework* – *Framework* para la edición gráfica) es un *plugin* para el desarrollo de editores visuales que pueden ir desde procesadores de texto hasta editores de diagramas UML, interfaces gráficas para el usuario, etc. Estos editores residen dentro del propio Eclipse por lo que pueden ser usados conjuntamente con otros *plugins*, ofreciendo una interfaz gráfica personalizable y profesional. GEF se utilizará para definir el metamodelo de forma gráfica y como soporte para el desarrollo gráfico con GMF.

En la Figura 4-11 se muestra el diagrama de clases UML 2.0 con las dependencias entre el editor gráfico generado para el DSL, el *runtime* de GMF, las herramientas EMF y GEF, y la plataforma Eclipse.

Para el desarrollo de editores gráficos para las modelos utilizando GMF son necesarios los siguientes pasos (véase la Figura 4-12):

- Crear un metamodelo (*Domain Model*) con la información no gráfica que gestionará el editor. Éste será el metamodelo de sintaxis abstracta del DSL, que se completará con reglas OCL para garantizar que las expresiones escritas basándose en el mismo sean válidas.
- Crear un modelo de definición del diagrama (*Diagram Definition Model*), donde se definen los elementos gráficos que se mostrarán en el editor. Este modelo define cómo se presentarán los elementos del diagrama al usuario (sintaxis concreta).
- Crear un modelo de correspondencias del diagrama (*Diagram Mapping Model*) que define las correspondencias entre los elementos gráficos y los del metamodelo. Estas correspondencias son esenciales para mantener sincronizadas las sintaxis abstracta (metamodelo) y la concreta (elementos gráficos). GMF mantiene sincronizadas ambas sintaxis de manera automática.
- Generar un editor gráfico a partir del modelo de correspondencias del diagrama.
- Mejorar y personalizar el editor gráfico editando el código del *plugin* que se ha generado.

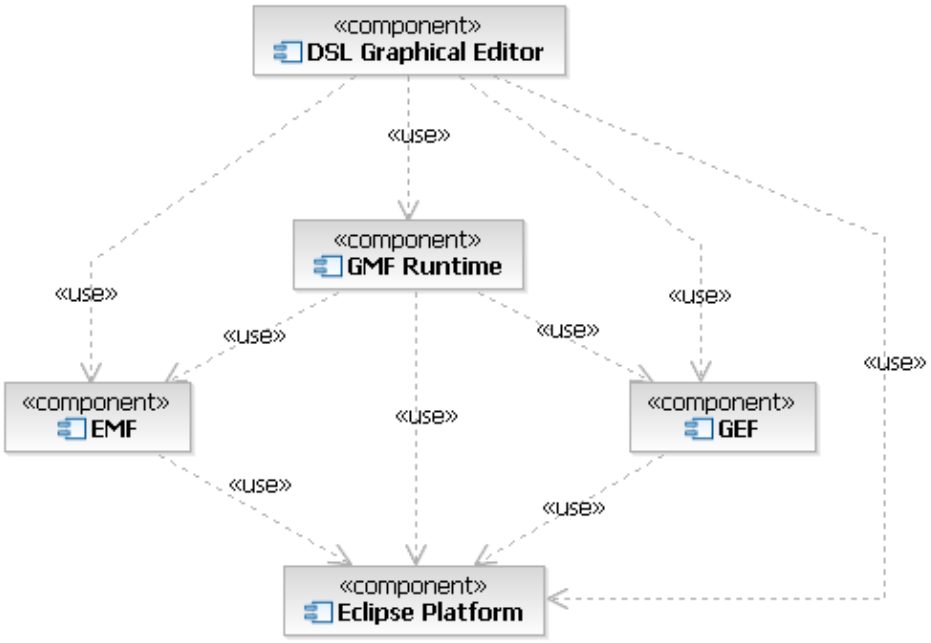


Figura 4-11. Dependencias entre las herramientas de modelado (GMF, EMF, GEF, y plataforma Eclipse). Fuente: [GMF 08].

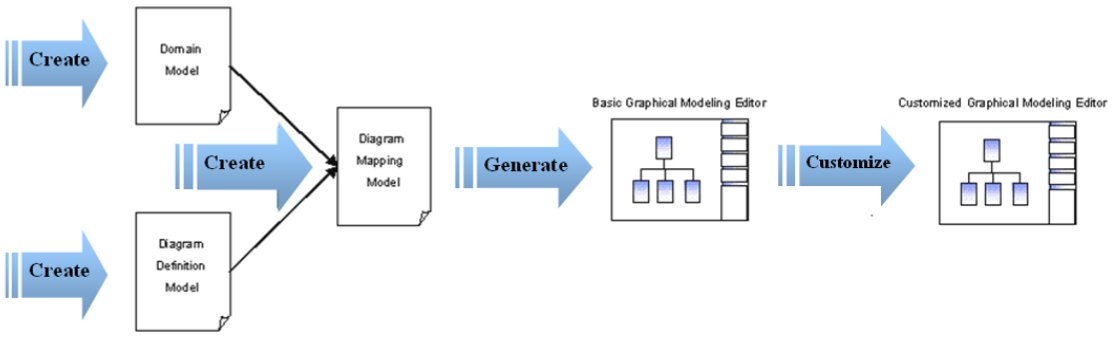


Figura 4-12. Flujo para la creación de un DSL gráfico con GMF. Fuente: [GMF 08].

El empleo de GMF para la creación del editor gráfico para el DSL aporta numerosas ventajas, entre las que cabe destacar las siguientes:

- Sus características proporcionan una interfaz consistente con otros editores gráficos basados en GMF.
- Los editores pueden crearse desde cero y/o ser generados por las herramientas desarrolladas como parte del SDK de GMF.
- Gestiona la persistencia del diagrama, permitiendo al desarrollador centrarse en la lógica de negocio.
- Es extensible flexible y abierto, lo que facilita su ampliación por desarrolladores externos.

- Está integrado con los nuevos componentes como OCL y otras herramientas de validación.
- Define un metamodelo de notación extensible para aislar la notación de los aspectos semánticos.
- Sus propiedades (denominadas *features*) están bien diseñadas, codificadas y verificadas.
- Las mejoras futuras a GMF son fácilmente integrables con los editores generados.

De la experiencia derivada de su uso se han detectado también algunos inconvenientes, como la dificultad en el aprendizaje, sensibilidad a cambios en el metamodelo, poca portabilidad entre versiones e inestabilidad tecnológica de la plataforma (se trata de una herramienta nueva que aún tiene que madurar). Esto ha implicado un esfuerzo adicional en aspectos como la creación de elementos gráficos personalizados o la realización de las sucesivas modificaciones para incluir mejoras en la herramienta.

4.5.1 Herramienta Desarrollada

Para el desarrollo del DSL propuesto en el apartado 4.4 (conforme al metamodelo presentado), se ha seguido el proceso de diseño descrito, haciendo uso de las herramientas disponibles dentro del entorno de desarrollo Eclipse.

El DSL permite la creación de modelos basándose en el manejo intuitivo de elementos gráficos, contribuyendo así a la sencillez en el desarrollo de aplicaciones domóticas. También se puede utilizar el modelo en formato textual (*Tree Editor*). Tanto la vista gráfica como la textual son consistentes entre ellas estando sincronizadas en todo momento.

El DSL gráfico desarrollado está integrado por cuatro secciones principales como se observa en la Figura 4-13 y se describe a continuación:

- Explorador: permite navegar el proyecto mediante las perspectivas disponibles en la herramienta Eclipse, la más utilizada será la perspectiva de "*paquetes*".
- Zona de dibujo: zona de trabajo donde se arrastran las figuras desde la paleta para construir un modelo gráfico concreto.
- Paleta de Primitivas gráficas: primitivas gráficas que se pueden arrastrar a la zona de dibujo. Una primitiva gráfica permite crear instancias de las definiciones gráficas, poseen propiedades o atributos que definen las características de sus figuras.
- Zona de propiedades: zona donde se visualizan y se modifican aquellas propiedades (atributos, parámetros, etc.) disponibles para la primitiva seleccionada. Para acceder a estas propiedades basta con seleccionar un elemento instanciado en la zona de dibujo. Las propiedades se muestran como campos editables o desplegables para facilitar un acceso rápido e intuitivo.

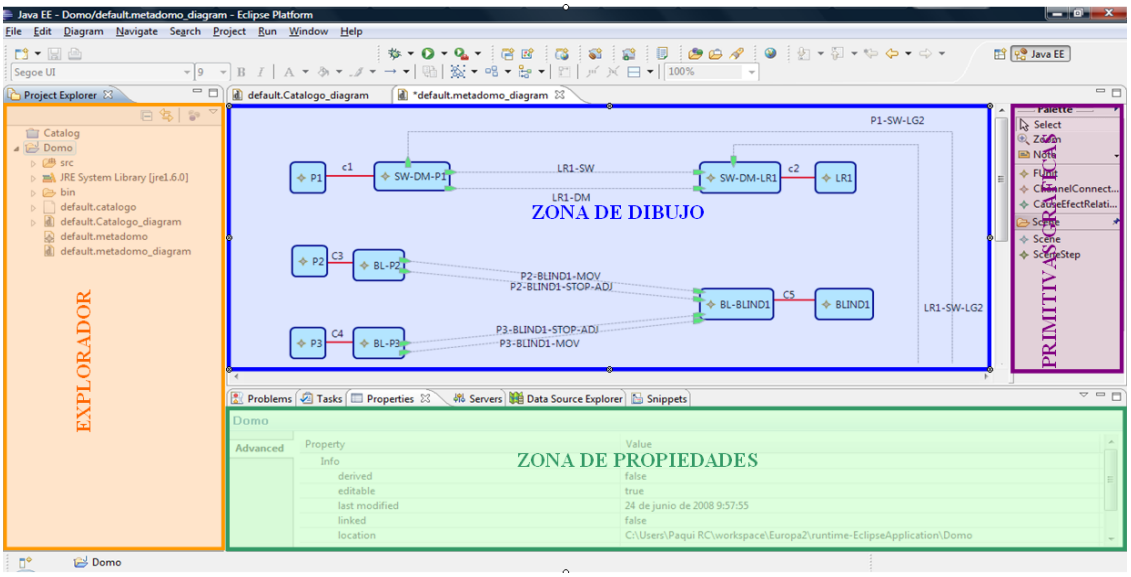


Figura 4-13. Captura de la herramienta para creación de aplicaciones con el DSL.

Utilizando el DSL desarrollado se pueden crear ejemplares de tantas unidades funcionales como sean necesarias. Las unidades funcionales están representadas de acuerdo con la sintaxis concreta definida anteriormente. El catálogo inicial (no acotado) de unidades funcionales permite instanciar pulsadores, persianas, puertas lógicas, sensores de presencia, pantallas de proyección, etc.

Así, se proporcionan todas las primitivas necesarias para construir modelos a partir de los conceptos definidos en la sintaxis abstracta (metamodelo) empleando la representación proporcionada por la sintaxis textual (elementos gráficos).

En el anexo A se describe una aplicación completa realizada con este DSL para el caso de estudio de una Sala de Reuniones.

4.6 Semántica

Todo lenguaje debe tener asociada una semántica que describa el significado de los conceptos que maneja. Además, la definición de la semántica en el contexto del enfoque dirigido por modelos es fundamental para definir las transformaciones y la ejecución y simulación de los modelos.

El DSL desarrollado emplea conceptos que ya tienen un significado bien definido en el contexto en que se utilizan (al menos para los expertos del dominio). Conceptos como pulsador, bombilla, motor tienen un significado por sí mismos. Por otra parte, también se proporciona una primera definición de semántica mediante las descripciones de las unidades funcionales, que son los elementos básicos del sistema domótico. Dichas descripciones se incluyen en el catálogo de unidades funcionales junto con sus parámetros y servicios.

La definición de la semántica se completará, de manera más formal, en niveles inferiores de la metodología, que se presenta en el capítulo 5, en el que se profundiza en el enfoque metodológico completo, donde este DSL se sitúa en el nivel de mayor abstracción (CIM: *Computer Independet Model*). En el nivel inmediatamente inferior (PIM: *Platform Independet Model*) se hará uso de un lenguaje de componentes que proporciona las herramientas para definir de manera precisa y formal el comportamiento de los componentes que maneja. De este modo, las unidades funcionales se transformarán a componentes cuyo comportamiento se describirá de forma precisa mediante características de este lenguaje (máquinas de estados finitos). Se utiliza por tanto un enfoque de semántica **traslacional** en el que los conceptos del DSL se trasladan a componentes de un lenguaje o modelo de nivel inferior. La traducción a estos componentes se realizará mediante trasformaciones de modelos, que se presentarán con detalle en el capítulo de la metodología.

4.7 Conclusiones y Aportaciones a la Tesis

En este capítulo se han presentado las características más importantes de los lenguajes específicos de dominio, partiendo de una revisión de los enfoques utilizados en múltiples campos hasta llegar a las propuestas existentes para el dominio de la domótica, analizando las ventajas e inconvenientes de cada uno de ellos. Además, se han revisado los elementos necesarios para definir un DSL: sintaxis abstracta, sintaxis concreta y semántica. Este estudio ha servido como punto de partida para definir un lenguaje específico del domino domótico con el mayor nivel de abstracción posible con una sintaxis concreta de tipo gráfico. Esta idea proporciona al usuario (que ahora no tiene que ser experto) un entorno gráfico donde definir los requisitos del sistema, quedando descrita toda la lógica de negocio desde una perspectiva independiente de computación e independiente de la plataforma. De este modo no se hace necesario comenzar el desarrollo analizando el software desde una perspectiva tecnológica.

Esta propuesta es el punto de partida para el desarrollo de la herramienta que dé soporte a una metodología más ambiciosa en el contexto MDA que permitirá definir las transformaciones de modelo a modelo necesarias desde el DSL, situado en un nivel independiente de la computación a un nivel intermedio de componentes que garantice la compatibilidad con otros sistemas reactivos como las redes de sensores, visión o la robótica. Finalmente se utilizarán herramientas de transformación de modelo a texto para completar el ciclo y obtener una serie de plantillas compatibles con las aplicaciones domóticas existentes.

En el capítulo siguiente se presentará la metodología que da soporte al ciclo de vida completo para el desarrollo de aplicaciones domóticas.

CAPITULO 5

Desarrollo de Sistemas Domóticos Dirigido por Modelos

En este capítulo se presenta HAbitATION (development of Home Automation ApplicaTIons using a mOdel driveN approach), una metodología para abordar el ciclo de vida completo de desarrollo de sistemas domóticos siguiendo un enfoque dirigido por modelos, basándose en la propuesta MDA del OMG, y en la utilización de lenguajes específicos de dominio como soporte para la definición de los requisitos de las aplicaciones. Con esta filosofía se pretenden solucionar los problemas asociados al proceso de desarrollo actual en el ámbito de la domótica. Entre otros, se agiliza el proceso desarrollo de aplicaciones, se facilita la verificación y se aumenta el nivel de abstracción en la definición de los requisitos del sistema. El éxito en los resultados obtenidos permitirá afianzar la adopción de técnicas y métodos de Ingeniería del Software en otras familias de sistemas reactivos con características y problemáticas similares.

El capítulo se organiza en cinco apartados. En primer lugar se expone la problemática asociada al proceso de desarrollo que se utiliza en la actualidad para sistemas domóticos. A continuación se propone la metodología basada en MDA que soluciona las deficiencias detectadas en la sección anterior. El apartado tres profundiza en las características propias de la plataforma KNX/EIB como tecnología de nivel PSM para obtener una implementación del sistema. Por último se describen las transformaciones necesarias para, a partir de los modelos de nivel CIM, obtener los modelos intermedios de componentes de nivel PIM y el modelo final de nivel PSM y se propone el enfoque a utilizar para la generación de código a partir de éste.

5.1 Proceso de Desarrollo Actual de Sistemas Domóticos

Tal y como se expone en capítulos anteriores, en la actualidad existe una amplia variedad de sistemas domóticos (propietarios o abiertos) entre los que destacan, por su penetración en el mercado, los sistemas recogidos en normas nacionales e internacionales Konnex (normas EN-50090 e ISO/IEC 14543-3-X) y Lonworks (normas ANSI EIA/CEA-709.1, IEEE 1473-L y EN-14908).

El desarrollo de aplicaciones con estas tecnologías se realiza empleando mayoritariamente las herramientas software proporcionadas bien por el fabricante de los dispositivos en el caso de sistemas propietarios, bien por las asociaciones encargadas de dar soporte a la tecnología en el caso de sistemas normalizados. Estas herramientas suelen ser entornos integrados dependientes de la plataforma y orientados a la generación de código, que elevan poco el nivel de abstracción y suelen proporcionar una sintaxis concreta poco intuitiva, siendo necesaria una formación muy especializada de los usuarios, que trabajan en un espacio muy cercano a la solución.

Los fabricantes de dispositivos domóticos proporcionan al instalador una serie de bibliotecas que se integran en los entornos de desarrollo y ofrecen la funcionalidad deseada. El instalador y/o administrador del sistema domótico se limitan a elegir los programas de aplicación de las bibliotecas y a fijar los parámetros de configuración. Las bibliotecas de un fabricante sólo funcionan con los dispositivos de ese fabricante, aunque es posible la interoperatividad con dispositivos de otros fabricantes siempre que sigan la misma normativa. El instalador y/o administrador de la red no puede modificar los programas de aplicación, sólo tiene acceso a la configuración de los parámetros de funcionamiento que define el fabricante. Además, las bibliotecas de un sistema que hace uso de una tecnología no son portables a otros. Por ejemplo, un programa de aplicación para KNX/EIB no es portable a LonWorks, aunque pueda existir una versión de programa de aplicación con una funcionalidad similar.

Las herramientas para la configuración de la red son propias de cada sistema: KNX/EIB utiliza ETS (*Engineering Tool Software*) y LonWorks utiliza LonMaker (con interfaz gráfica basada en *Microsoft Visio*) para la definición de la red (véase la Figura 5-1.). Aunque la filosofía de funcionamiento de los estándares más empleados es similar, **no se conoce la existencia de herramientas con el suficiente nivel de abstracción para modelar el sistema domótico a alto nivel y obtener una implementación en uno u otro sistema de manera automática**.

En definitiva, para el desarrollo de aplicaciones se utilizan herramientas propias de cada sistema. Estos paquetes de software son comercializados por las organizaciones encargadas de la normalización del protocolo y la certificación de productos (*Echelon* para Lonworks y *Konnex Association* para KNX/EIB). El proceso seguido es el siguiente:

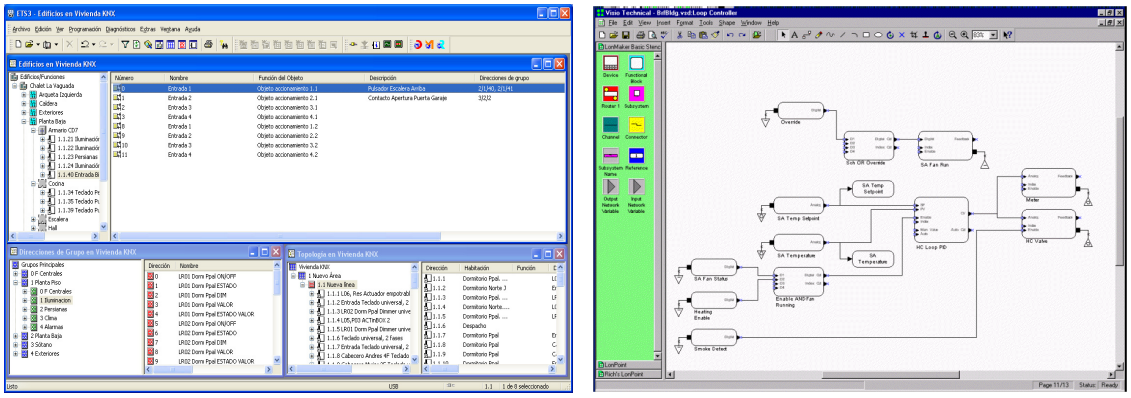


Figura 5-1. Herramientas ETS (*Engineering Tool Software*) y LonMaker para las tecnologías Konnex y Lonworks, respectivamente.

- Definición de los requisitos de la instalación.
- Elección del sistema o tecnología domótica. En instalaciones residenciales no se suele integrar más de un sistema. En grandes edificios se puede plantear comunicar redes domóticas con tecnologías diferentes en función de las necesidades de la instalación.
- Planificación y proyecto de la instalación: selección de material y tipo de dispositivos que se van a emplear.
- Realización de la instalación eléctrica, dirigida por los ingenieros encargados del proyecto.
- Programación de los dispositivos con las herramientas de configuración. La programación consiste en seleccionar una serie de programas de aplicación (o *drivers*) del catálogo que proporciona el fabricante de los dispositivos y a su parametrización (con las limitaciones expuestas anteriormente).
- Verificación del funcionamiento de la instalación (no es posible realizar simulaciones previas a la programación).

Todo el proceso es realizado por un especialista del dominio (ingeniero de proyecto), que recoge los requerimientos del cliente para una instalación (elementos a integrar, funcionalidad requerida, elección de una tecnología concreta) basándose en su propia experiencia, realiza la instalación, coordina el seguimiento y por último debe programar los dispositivos para conseguir la funcionalidad. Con esta forma de trabajar resulta difícil satisfacer algunos atributos deseables en el desarrollo de sistemas software [Sommerville 00]:

- Interoperabilidad: la variedad de estándares existentes y la falta de uno común ha empujado a los fabricantes a desarrollar sus propios sistemas, muchas veces cerrados, haciendo así muy complicado la interoperabilidad entre ellos. Este mismo problema afecta a la oferta de dispositivos soportados por un estándar específico, imposibilitando trabajar con distintos sistemas y protocolos. En muchos casos el sistema final también se ve limitado por la oferta de dispositivos que soporten un determinado estándar.

- Flexibilidad: el cambio en el software es una consecuencia inevitable de un cambio en el entorno de negocio y, dada la constante evolución de los sistemas domóticos, debe ser un objetivo básico para el desarrollo de estos sistemas ofrecer el soporte necesario para afrontar estos cambios. El problema surge debido a que los sistemas domóticos se definen con conceptos don bajo nivel de abstracción, lo que en algunos casos complica la modificación o ampliación del sistema, convirtiéndose en un problema de escalabilidad.

- Robustez: normalmente el desarrollo de la aplicación lo hace el especialista una plataforma específica, que utiliza conceptos muy dependientes de la tecnología, lo que incrementa la probabilidad de que se produzcan errores en el software. Esto dificulta la realización de cambios sin que afecten al resto del sistema, especialmente cuando la aplicación es compleja.
- Reutilización: los sistemas domóticos cuentan con una serie de dispositivos y funcionalidades que se repiten en todas las instalaciones. Sin embargo, con el proceso de desarrollo seguido actualmente es necesario realizar toda la implementación de principio a fin cada vez que se desarrolla. Algunos fabricantes cuentan con bibliotecas que ayudan a la reutilización, pero no permiten un aprovechamiento sistemático del software, sobre todo en las primeras etapas de desarrollo.

- Productividad: el incumplimiento, en mayor o menor grado, de los atributos anteriores repercute en una baja productividad y un nivel de calidad que es muy dependiente de la experiencia y formación del experto del dominio. Para el desarrollo de estos sistemas se requiere personal altamente especializado en cada una de las plataformas. Por otro lado, para cambiar un sistema de plataforma es necesario desarrollar todo el sistema de nuevo para esa plataforma. Además, la validación del sistema se realiza en las últimas etapas del desarrollo lo que implica que para una rectificación se deba desarrollar el sistema desde el principio, etc. Todos estos problemas hacen que el desarrollo se alargue en el tiempo, los costes aumenten y la productividad se vea degradada.

En resumen, el proceso actual de desarrollo presenta numerosos inconvenientes y no satisface los principios básicos de la Ingeniería del *Software*. Los motivos que ocasionan dichos problemas se pueden resumir en los siguientes puntos:

- No existe una metodología rigurosa ni sistemática para recoger los requerimientos de la aplicación (se suelen utilizar formularios).
- Las decisiones en el proceso de desarrollo están condicionadas por la experiencia del diseñador, que suele utilizar la tecnología en la que tienen más experiencia y las soluciones que conoce.
- El nivel de abstracción en la especificación de requisitos es bajo, ya que se realiza directamente sobre una tecnología y no con conceptos del dominio.

- Las herramientas disponibles son específicas para una tecnología y en la mayoría de los casos son poco intuitivas y dirigidas a diseñadores expertos.

En los siguientes apartados se presentará una metodología basada en el desarrollo dirigido por modelos y el empleo de lenguajes específicos de dominio para abordar el desarrollo de sistemas domóticos utilizando abstracciones propias de este dominio, a partir de las cuales se podrá obtener de forma automática o semiautomática el código para diferentes plataformas o tecnologías de implementación.

5.2 Metodología Propuesta

Dada la problemática que presenta el desarrollo tradicional de software para sistemas domóticos, se hace necesaria una metodología que resuelva las deficiencias descritas en el apartado anterior y mejore tanto la calidad como la productividad en el proceso de desarrollo. Dicha metodología debe recoger los requisitos del sistema domótico independientemente de la plataforma o estándares que se vaya a emplear en la implementación, facilitando así la portabilidad entre sistemas. Además, las herramientas desarrolladas deben ser intuitivas y de fácil manejo (no requerir formación especializada en los diferentes sistemas), permitir la reutilización de los elementos software y facilitar la extensibilidad de las aplicaciones.

Con estos objetivos se propone HAbitATION (*development of Home Automation ApplicaTIons using a mOdel driveN approach*), una metodología que utiliza el enfoque MDE para dar soporte completo al ciclo de vida del desarrollo de sistemas domóticos, desde la captura de requisitos hasta la implementación en una plataforma específica. MDE hace uso, de forma sistemática y reiterada, de modelos como elementos básicos a lo largo de todo el proceso de desarrollo [Kent 02] y se consigue, entre otros, la separación del modelo de negocio de la plataforma de implementación, la identificación, expresión precisa, separación y combinación de aspectos específicos del sistema en desarrollo con DSLs, el establecimiento de relaciones precisas entre los diferentes lenguajes en una arquitectura global y, en particular, la posibilidad de expresar transformaciones entre ellos [Bézivin 05a]. Todos estos aspectos van a ser abordados en mayor o menor medida a lo largo del capítulo.

En este trabajo de Tesis se utiliza la propuesta MDA del enfoque MDE, realizada por el OMG, que organiza el desarrollo de software en tres capas, y a su vez contempla tres familias de modelos en función del nivel de abstracción: modelo independiente de la computación (CIM), modelo independiente de la plataforma (PIM) y modelo específico de la plataforma (PSM). Además, se propone el empleo de un lenguaje específico de dominio para la captura de requisitos del sistema (véase la Figura 3-11 y la Figura 3-12 del apartado 3.4).

El diagrama de la Figura 5-2 resume la metodología propuesta siguiendo la organización de capas MDA. En la capa CIM se realiza la captura de requisitos mediante un lenguaje específico de dominio. Los modelos de requisitos son

transformados a un lenguaje de componentes en la capa PIM y, mediante nuevas transformaciones, los modelos de componentes se traducen a código ejecutable en las diferentes plataformas de destino. En todas las etapas se han definido metamodelos para la creación de modelos conforme a ellos y las transformaciones necesarias para llegar hasta la generación de código.

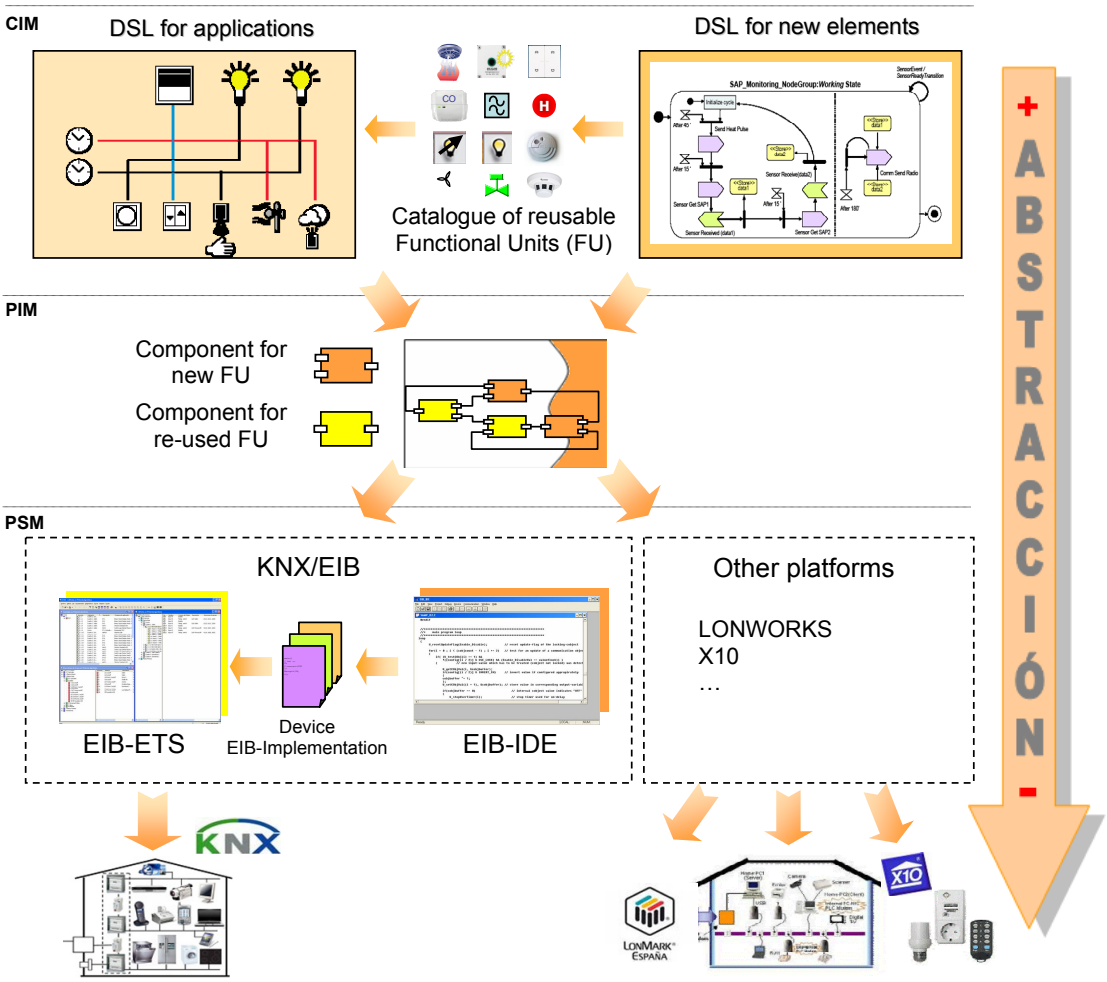


Figura 5-2. Metodología Propuesta.

5.2.1 Nivel Independiente de la Computación (CIM)

El diseño de las aplicaciones comienza, en la capa CIM, con la captura de requisitos con un alto nivel de abstracción (parte superior de la Figura 5-2), que se realiza mediante lenguajes de modelado cuya sintaxis abstracta es definida por metamodelos.

La especificación de requisitos permite a los diseñadores desarrollar un sistema que cumpla con las necesidades reales de los usuarios, incrementando de esta manera el éxito del proyecto. Esta tarea no es trivial, ya que el análisis de requisitos está relacionado con la comprensión, por parte del desarrollador, de procesos de toma de decisiones y los objetivos que el usuario pretende alcanzar con la información suministrada. Por lo tanto, obtener directamente del usuario requisitos de información es una tarea compleja, por lo que se requiere de técnicas especiales. Es por esto por lo

que se propone una aproximación basada en modelos mediante el uso de un DSL gráfico para definir requisitos de información con una notación gráfica cercana a la forma de pensar de los expertos del dominio y en la que no sean necesarios más conocimientos que los propios del dominio domótico.

El DSL hace uso de una serie de elementos domóticos que se encuentran disponibles en un **catálogo** con elementos reutilizables. En este nivel también se contempla la posibilidad de utilizar elementos que no existan en el catálogo (zona derecha de la Figura 5-2). Se puede diseñar un **nuevo elemento** definiendo su comportamiento para ser utilizado posteriormente en el desarrollo de la aplicación domótica. Además, este nuevo elemento se añade al catálogo para que pueda ser reutilizado en otros desarrollos. Esta parte de la metodología queda fuera del ámbito de este trabajo de Tesis, pero está prevista su realización como trabajo inmediato en el seno del Grupo de Investigación DSIE.

Para la definición del DSL de diseño de aplicaciones siguiendo el enfoque MDA se hace uso de un metamodelo que le da soporte. En el capítulo 4 se describieron con detalle todos los elementos que conforman la definición del lenguaje: sintaxis concreta (basada en diagramas gráficos), el metamodelo para la definición de su sintaxis abstracta, las reglas OCL para garantizar la corrección de las expresiones del lenguaje y el tipo de semántica para darle significado. Basándose en el mismo metamodelo, se ha definido un segundo DSL, mucho más sencillo, para la creación y mantenimiento del catálogo de componentes reutilizables. Asimismo, el anexo A ilustra su utilización en un caso de estudio.

Una vez definidos los requisitos de una aplicación domótica es necesario avanzar hacia un modelado funcional de la aplicación. Para ello debe transformarse el modelo obtenido con el DSL en el nivel CIM en un modelo intermedio de nivel PIM basado en componentes. El nivel PIM (parte intermedia de la Figura 5-2) especifica el sistema mediante componentes manteniendo una independencia de la plataforma. La especificación se formula haciendo uso de un conjunto de modelos que representan una visión orientada a componentes de los elementos utilizados en la fase anterior.

5.2.2 Nivel Independiente de la Plataforma (PIM)

En la capa PIM, al igual que en la fase anterior, se distingue entre componentes que representan elementos catalogados (representados con fondo amarillo en la Figura 5-2) y aquellos que representan elementos de nueva creación (con fondo naranja). Para estos últimos habrá que especificar su comportamiento haciendo uso de máquinas de estado y diagramas de actividad fundamentalmente. Una de las motivaciones para introducir este nivel en la metodología es que permite la confluencia entre diferentes dominios de sistemas reactivos (robótica, redes de sensores inalámbricos, visión artificial, etc.). De hecho, ya existen algunos trabajos como los de F. Losilla [Losilla 07a][Losilla 07b] o D. Alonso [Alonso 08b] que lo utilizan para el desarrollo de redes de sensores y aplicaciones robóticas respectivamente. Así, se dispone de un conjunto de

componentes reutilizables que pueden ensamblarse con componentes de otros dominios manteniendo la compatibilidad entre ellos.

La definición formal de un sistema utilizando una arquitectura de componentes, desde un punto de vista independiente de la plataforma, debe contener diferentes vistas. Para afrontar con éxito una descripción sin ambigüedad, legible y adecuada del sistema, B. M. Duc propone tres vistas o perspectivas básicas [Duc 07]:

- Vista estructural. Describe un sistema como una colección de objetos o componentes, sus características y relaciones.
- Vista funcional. Define la capacidad potencial de las acciones que pueden realizar las clases o componentes del sistema.
- Vista dinámica. Representa cómo se implementa la funcionalidad recogiendo aspectos temporales, comportamentales y evolutivos del sistema. Complementa a la vista funcional y determina los estados de cada componente del sistema, eventos y condiciones que conforman el comportamiento, colaboración entre acciones para conseguir un objetivo, etc. En UML se utilizan diagramas de actividad, secuencia, interacción y de máquinas de estado.

Para el nivel PIM existen diferentes alternativas a la hora de escoger el modelo de componentes. Una opción es utilizar un modelo de componentes de propósito general de entre los muchos existentes. Los más importantes son JavaBeans [Tratt 05], EJB [DeMichiel 06] [Hamilton 97] de *Sun Microsystems*, .NET y COM de Microsoft, CCM (*Corba Component Model*) de OMG [OMG 04], Koala [Ommerging 00], SOFA [Plasil 98], KobrA [Atkinson 01], ADLs [Clements 96], UML2 [UML v2.1.1 07], PECOS [Nierstrasz 02], Pin [Ivers 02] y Fractal [Bruneton 03]. Estos modelos de componentes se pueden clasificar desde diferentes puntos de vista: según su sintaxis, su semántica o su capacidad para la composición. Desde las perspectivas sintáctica y semántica existen dos grandes grupos [Lau 05]:

- Los basados en lenguajes de programación (orientados a objetos o apoyados por lenguajes de definición de interfaces), que utilizan clases u objetos, como JavaBeans, EJB, COM, CCM y Fractal.
- Los basados en lenguajes de descripción de arquitecturas que utilizan unidades arquitecturales, como ADLs, UML2.0, KobrA, Koala, SOFA, PECOS y Pin.

El principal problema radica en la falta de consenso entre los diferentes modelos de componentes de propósito general, además de que muchos de ellos (JavaBeans, EJB, COM, CCM, Fractal) tratan los componentes como simple código fuente o binario. Como afirma Kang en su estudio [Kang 06] "la reusabilidad debe ser una característica inherente al software", algo que no se puede lograr, como se ha demostrado hasta ahora, desde el nivel del código fuente, por muy bien diseñado que esté. Tal como se afirma en [Ortiz 05], es necesario que el concepto de componente sea independiente de la plataforma final de ejecución. El enfoque MDE se perfila como la tecnología perfecta para tratar el componente de forma totalmente independiente de la tecnología de

implementación final. En esta Tesis Doctoral se defiende que la verdadera reutilización se va a conseguir cuando la unidad de reutilización sea el modelo, en vez de directamente el código fuente.

Por ello, siguiendo estas premisas, los modelos de componentes más adecuados a la filosofía promulgada por MDE son los de tipo arquitectural. Para este trabajo de Tesis se ha optado por el empleo de un nuevo **modelo de componentes creado en el seno del Grupo DSIE específicamente para el desarrollo software de sistemas reactivos**. La principal ventaja de este modelo de componentes, denominado V³Studio, frente a otros más genéricos, como UML, radica en su simplicidad y facilidad de uso. A diferencia de UML, todos los elementos del metamodelo de V³Studio están claramente definidos y diferenciados, y no dispone de formas alternativas para modelar la misma realidad manteniendo, en la medida de lo posible, la capacidad expresiva y de modelado suficiente para desarrollar el esqueleto completo de un sistema del dominio. Además, V³Studio ya ha sido verificado con éxito en el campo de la robótica [Alonso 08b] y las redes de sensores inalámbricas [Losilla 07a][Losilla 07b], por lo que a la vez **facilita la confluencia entre diferentes dominios**.

V³Studio se ha desarrollado a partir de un metamodelo inspirado en UML siguiendo las premisas básicas del enfoque dirigido por modelos. Además, para su diseño se ha utilizado la propuesta MDA y el entorno Eclipse, por lo que su integración en la metodología propuesta resulta perfectamente viable desde el punto de vista de la compatibilidad de las herramientas utilizadas para definir las transformaciones de modelos que guían el proceso de desarrollo.

V³Studio contempla tres vistas que permiten modelar cada uno de los tres aspectos de un sistema basado en componentes (véase la Figura 5-3). Aunque con diferente denominación y algunas particularizaciones, cada una de estas vistas define uno de los tres aspectos básicos que se describieron anteriormente y que conforman un sistema basado en componentes:

- Vista arquitectónica: permite definir los componentes (ya sean simples o compuestos) que forman la aplicación, sus puertos, los tipos de datos y las interfaces globales del sistema, los servicios ofrecidos por las interfaces, asociar al componente una máquina de estados y conectar componentes entre sí.
- Vista de comportamiento: permite definir una máquina de estados que describa tanto el comportamiento interno de un componente como su reacción ante la solicitud, por parte de otros componentes, de alguno de los servicios que proporciona.
- Vista algorítmica: permite describir la secuencia de algoritmos que van a ser ejecutados cuando un componente se encuentre en un estado concreto o como respuesta a la solicitud de alguno de los servicios ofrecidos. Estos algoritmos son ejecutados únicamente cuando el componente se encuentra en el estado adecuado.

Si se compara con la propuesta de M. Duc [Duc 07], la vista arquitectónica recoge la estructura del sistema y, para cada componente, el conjunto de puertos (e interfaces) de que dispone, por lo que se recogen también los aspectos funcionales básicos de los componentes (servicios y operaciones que son capaces de realizar). Los aspectos de la vista dinámica se formalizan en las vistas de comportamiento y algorítmica). Las vistas propuestas en V³Studio se han diseñado para recoger de manera precisa y completa las características de los sistemas reactivos, como ha demostrado su aplicación a los dominios de la robótica y las redes de sensores.

El metamodelo de V³Studio permite desarrollar cada una de las partes que definen un aspecto de la arquitectura del sistema por separado, ya que la unicidad de su metamodelo asegura la unicidad de las mismas cuando se unen. Para ello, al igual que el metamodelo propuesto para el nivel CIM, cuenta con una serie de restricciones OCL que aseguran que el modelo está bien formado y es correcto.

Finalmente, el modelo de componentes V³Studio aporta un elemento esencial en esta metodología, ya que provee un soporte para la semántica de los componentes mediante máquinas de estado y diagramas de actividad. Puesto que se ha empleado un enfoque semántico traslacional en el nivel CIM, se aprovecha la disponibilidad de estos recursos en el nivel PIM para dotar de significado a los dispositivos domóticos utilizados en las aplicaciones, especialmente los de nuevo diseño.

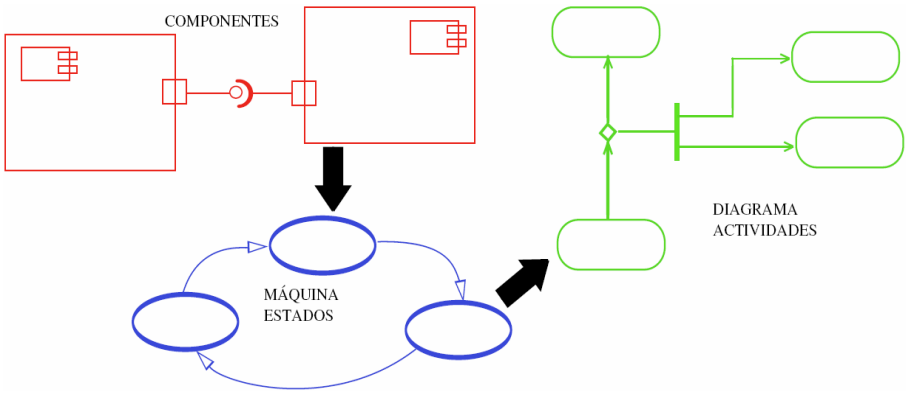


Figura 5-3. Esquema del metamodelo de V³Studio. Fuente: [Alonso 08a].

5.2.3 Nivel Específico de la Plataforma (PSM)

La metodología propuesta completa el ciclo de vida en el desarrollo de un sistema domótico con la generación de código de forma automática (o semiautomática) para una plataforma específica. Para llevar a cabo esta tarea es necesario conocer en profundidad las características del sistema domótico o plataforma para la que se desea generar la aplicación, así como disponer de las herramientas (compiladores, entornos de configuración, etc.) necesarias para tal fin. En función de la complejidad del sistema esta tarea puede ser más o menos laboriosa. En el siguiente apartado se describen con más detalle las consideraciones necesarias para abordar esta última fase del desarrollo.

5.3 Nivel Específico de Plataforma (PSM). Tecnología KNX/EIB

La etapa final en el desarrollo de un sistema domótico consiste en la selección y programación de los dispositivos para cumplir con los requisitos impuestos en las primeras etapas del proceso. Para ello es necesario seleccionar una plataforma sobre la que realizar la implementación, con el fin de que los dispositivos escogidos funcionen de acuerdo con un mismo protocolo. En el capítulo 2 se describieron las principales tecnologías domóticas existentes en la actualidad junto con sus características más importantes. De este estudio se pueden extraer varias conclusiones importantes a la hora de seleccionar la plataforma de destino:

- Existe una gran variedad de sistemas domóticos, algunos de ellos estandarizados y otros no.
- Es conveniente utilizar una tecnología abierta, que se encuentre recogida en normas nacionales e internacionales para garantizar la disponibilidad de productos de distintos fabricantes a medio y largo plazo, así como la compatibilidad entre ellos.
- Los estándares con mayor implantación en el ámbito europeo e internacional son Lonworks y Konnex, y en particular su especificación EIB sobre bus de par trenzado (especificación KNX/TP1).

Tanto Lonworks como KNX/EIB se basan en el modelo de referencia OSI de la ISO, y definen protocolos para las diferentes capas de este modelo abierto para interconexión de sistemas. Por ello, es fácil deducir que la creación de herramientas para la generación de aplicaciones de forma automática para cada una de las plataformas no es algo trivial, y requiere un profundo conocimiento de cada una de ellas. Por lo tanto, en este trabajo de Tesis se ha restringido el alcance a la definición de los modelos y transformaciones del nivel PSM a una de las plataformas domóticas de mayor relevancia: el sistema KNX/EIB en su implementación TP1 (más del 90% de los productos KNX utilizan el medio TP1 [Kyselytsya 06]).

La creación de una aplicación KNX/EIB dentro del marco MDA tiene una serie de condicionantes específicos que lo hacen diferente a otros sistemas software en los que se genera código ejecutable en algún lenguaje de propósito general (Java, C++, Ada, etc.). Estos condicionantes están relacionados con la forma de trabajar con la tecnología KNX/EIB. En el capítulo 2 se describieron sus principales características, y en el capítulo 4, dedicado a los DSLs, se presentaron algunos detalles de la herramienta de empleada por los especialistas del dominio (ETS: *Engineering Tool Software*) para la programación de los dispositivos de este sistema. No obstante, para comprender mejor la metodología que se propone se revisarán los conceptos que tienen una mayor repercusión en el proceso de la generación de modelos y código ejecutable para las aplicaciones con esta plataforma.

5.3.1 Conceptos del Dominio KNX/EIB

En KNX/EIB se trabaja con dispositivos comerciales, comercializados por numerosos fabricantes. La compatibilidad entre dispositivos está garantizada por la asociación Konnex, que establece los procedimientos para conceder el marcado KNX. Este marcado garantiza la compatibilidad tanto de los elementos *hardware* como *software* con las especificaciones de esta tecnología. Existe un amplio abanico de dispositivos para cubrir todas las necesidades en instalaciones domóticas para viviendas y edificios. El fabricante proporciona junto con el dispositivo una librería que contiene el software que se ejecutará en el mismo, y que se parametriza y descarga desde una herramienta de integración, proporcionada por la asociación Konnex, que permite realizar la programación de cualquier dispositivo, independientemente de cual sea su fabricante.

Cada dispositivo incluye un microcontrolador, la electrónica de acondicionamiento necesaria para el tratamiento de las señales de entrada/salida para interactuar con el entorno, y una interfaz de comunicaciones según el protocolo KNX/EIB. Se trata, por tanto, de una red de comunicaciones distribuida, cuya topología se ha descrito en el capítulo 2. En la Figura 5-4 se muestra un ejemplo simplificado con dos dispositivos KNX/EIB y los conceptos más importantes implicados en la configuración de la instalación.

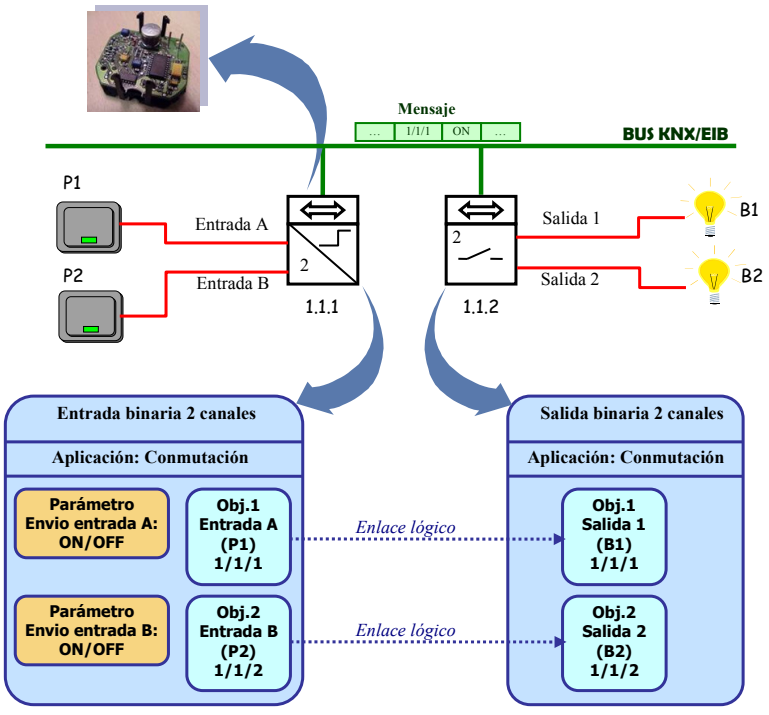


Figura 5-4. Ejemplo de conceptos del dominio KNX/EIB.

Cada dispositivo se identifica con una **dirección física** (por ejemplo 1.1.1) que es única y permite direccionarlo para su programación y posterior verificación del funcionamiento del sistema mediante monitorización del bus de datos. En el ejemplo de la Figura 5-4 se muestra un dispositivo de dos entradas binarias (con dirección 1.1.1)

y uno de dos salidas (con dirección 1.1.2). Estos elementos se han utilizado para conectar pulsadores y bombillas en sus canales de entrada y salida respectivamente.

Además, en cada dispositivo se carga un **programa de aplicación** que va a determinar su comportamiento. De esta manera, un dispositivo puede realizar múltiples funciones, dependiendo del programa que se cargue en cada momento. El programa de aplicación incluye dos elementos básicos: **parámetros** para configurar la forma en que funciona y **objetos de comunicación** para la realización de asociaciones o enlaces lógicos con otros elementos de la instalación. En este ejemplo, la entrada binaria dispone de dos objetos de comunicación, uno para cada uno de los canales de entrada, y dos parámetros que determinan el comando que se envía por el bus en caso de detección de un cambio en la señal de entrada de cada canal. La salida binaria dispone asimismo de dos canales asociados a relés que proporcionan tensión a dos bombillas (B1 y B2).

Las asociaciones lógicas se realizan mediante **direcciones de grupo**. Estas direcciones son creadas libremente por el experto del dominio que las asocia a objetos de comunicación de los dispositivos para enlazarlos entre sí. El formato utilizado es de tres números separados por barras (1/1/3). Así, al objeto de comunicación de la entrada A del dispositivo "Entrada Binara 2 canales" se le ha asociado la dirección de grupo 1/1/1, la misma que al objeto de la salida 1 del actuador "Salida binaria 2 canales", de modo que se establece un enlace lógico o virtual entre ambos objetos de comunicación. Por lo tanto, cuando se produzca una pulsación en P1, el objeto 1 de la entrada A desencadenará el envío de un mensaje de difusión en el bus (denominado telegrama en KNX/EIB) que contendrá, entre otros campos, la dirección de grupo de destino (1/1/1) y un comando de ON u OFF (encendido o apagado). Dicho mensaje será recibido por todos los dispositivos conectados al bus, pero sólo aquellos con la dirección de grupo contenida en el mensaje (1/1/1) lo procesarán. Así, el actuador "Salida binaria 2 canales" identificará esta dirección de grupo y activará (o desactivará según el caso) el relé de su salida 1, que producirá el encendido (o apagado) de la bombilla B1. Lo mismo ocurriría para la entrada B y la salida 2, asociadas mediante la dirección de grupo 1/1/2.

Los objetos de comunicación tienen un tipo de datos (los tipos de datos están normalizados en la especificación de la norma como DPTs o *Data Point Types*), de forma que sólo es posible enlazar mediante direcciones de grupo objetos de comunicación compatibles entre sí (esto es, que tengan el mismo tipo). Existen tipos de datos normalizados para ejecutar acciones que van desde una simple conmutación al envío de órdenes de movimiento de motores o valores en punto flotante para control de temperatura, humedad y otras magnitudes físicas.

El ejemplo descrito se ha simplificado para ilustrar de forma comprensible los conceptos utilizados en el dominio. Una instalación real implica la utilización, según su tamaño, de decenas o incluso cientos de dispositivos, cada uno de los cuales puede disponer de varios canales de entrada o salida, por lo que el diseño y programación de un sistema domótico puede ser una tarea compleja, en la que la facilidad de uso de las herramientas y la reutilización de soluciones a problemas habituales son factores

importantes. Por ejemplo, soluciones ya diseñadas para la gestión de cargas o el control de la climatización en una estancia no son fácilmente trasladables entre diferentes proyectos con funciones del tipo "copiar y pegar", puesto que los dispositivos del proyecto origen y destino no tienen por qué ser del mismo fabricante, además de los conflictos que se producirían entre las asignaciones de direcciones de grupo, que pueden estar ya en uso en el sistema de destino.

5.3.2 Enfoque Propuesto para la Generación de Código

La programación de una instalación KNX/EIB es realizada habitualmente por un experto del dominio (experto en domótica, pero además experto en esta tecnología) utilizando la herramienta de integración ETS o *Engineering Tool Software* en el modo de funcionamiento *System Mode* (véase la Figura 5-5), que es el único que permite una configuración completa de todos los dispositivos de la instalación. Esta herramienta, presentada en el apartado 4.3.2, incluye las funciones necesarias para insertar dispositivos en la instalación, fijar sus parámetros, descargar la programación a través del bus en los aparatos y realizar diagnósticos de funcionamiento.

En la Figura 5-6 pueden observarse diferentes capturas del entorno ETS para la configuración de los dos dispositivos descritos anteriormente (véase la Figura 5-4). Para la creación del proyecto han de definirse tres vistas (topología, edificios y direcciones de grupo), que se mantienen sincronizadas en todo momento. La parametrización se realiza mediante cuadros de diálogo accesibles seleccionando el dispositivo cuyos parámetros se desean modificar. La utilización de vistas a modo de árbol (no gráficas) dificulta considerablemente el proceso de diseño puesto que las asociaciones existentes entre objetos se hacen mediante direcciones de grupo, que se identifican, como ya se ha indicado, por un número (por ejemplo 1/1/2). La mecánica para añadir más funciones al sistema sería la descrita: insertar nuevos aparatos a partir de las librerías del fabricante, fijar los valores deseados para los parámetros y crear asociaciones mediante nuevas (o existentes) direcciones de grupo.

Con esta filosofía de trabajo, que se basa en una interacción manual del experto del dominio con la herramienta, surge el problema de cómo abordar la programación del sistema en el ámbito de la metodología dirigida por modelos presentada en esta Tesis. El objetivo es conseguir, mediante transformaciones modelo a modelo y modelo a texto, la generación automática o semiautomática de código que permita programar un sistema domótico en esta plataforma. Para ello se pueden plantear dos enfoques alternativos:

- Utilizar *plugins* de macros o *scripts* que se pueden integrar en ETS y permiten acceder a todo el sistema de objetos y realizar todas las tareas que se hacen de forma manual mediante alguno de los lenguajes de soportados (Java, XML, VBScript, etc.).

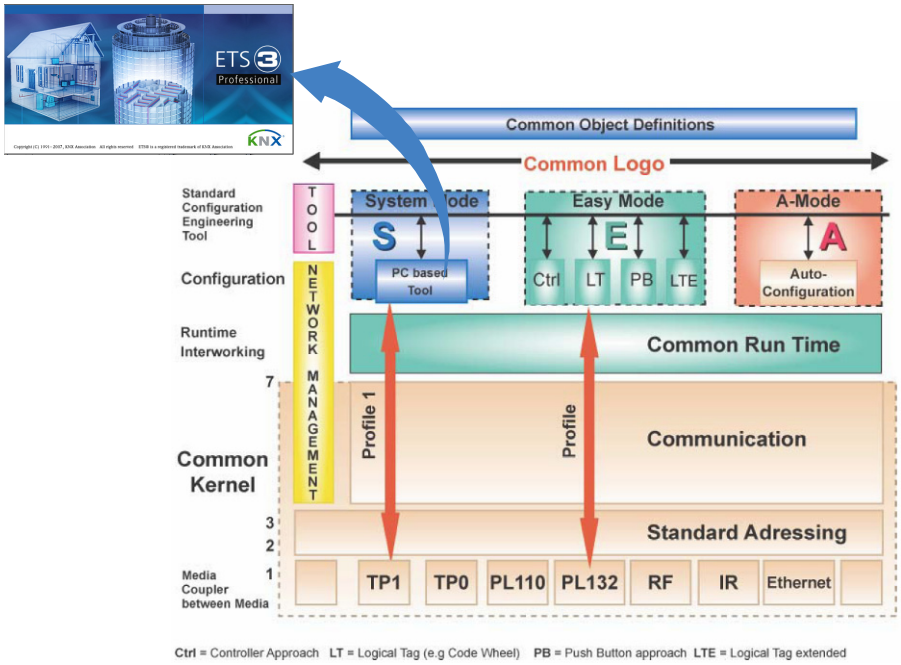
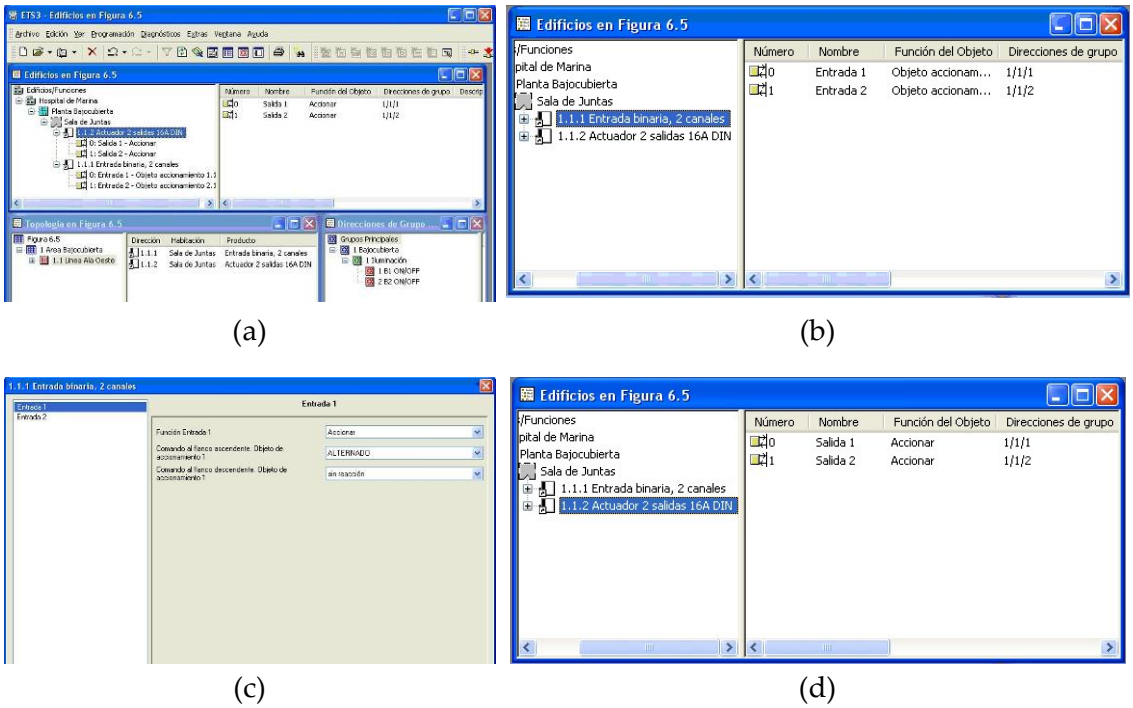


Figura 5-5. Modelo de la red KNX. Fuente: [KNX 04].



(a) (b)

(c) (d)

Figura 5-6. Programación del ejemplo de la Figura 5-4 con ETS. (a) Captura del entorno ETS. (b) Asignación de direcciones de grupo a los objetos de comunicación de la entrada binaria. (c) Parámetros de la entrada binaria. (d) Asignación de direcciones de grupo a la salida binaria.

- Diseñar el software necesario para sustituir al entorno ETS para la creación de aplicaciones KNX/EIB (TP1). Esto supondría realizar una tarea importante de ingeniería inversa para extraer la información contenida en las bases de datos de los fabricantes, puesto que no son abiertas, y la creación de un entorno

adecuado para su gestión. Esto es lo que se ha intentado en la universidad de Viena con la herramienta libre BASYS[1] [Kastner 05], con resultados no del todo satisfactorios: la herramienta no es estable y no es capaz de leer correctamente todas las bases de datos. No obstante, si evoluciona y madura puede ser una alternativa muy interesante, ya que trabaja con ficheros XML tanto para la descripción de los programas de aplicación, parámetros y objetos de las librerías del fabricante, como para los proyectos creados. Esto facilitaría en gran medida su manipulación dentro del enfoque dirigido por modelos, ya que partiendo de los esquemas XML que se utilizan para generar estos ficheros se podrían extraer sus metamodelos para así definir transformaciones M2M desde el nivel PIM.

La primera alternativa es la única viable por el momento, y se ha utilizado con éxito en [Shehata 07] para la definición y análisis de políticas en sistemas domóticos KNX/EIB. Por tanto, en la metodología propuesta se utilizará un *plugin* de macros para la generación de código en la plataforma KNX/EIB.

Para comprender mejor el enfoque utilizado es conveniente analizar el modelo de la arquitectura *software* de la plataforma de destino. ETS está construido sobre un *framework* de componentes según el modelo DCOM (*Distributed Component Object Model*, componentes de Ingeniería del *Software* para plataformas PC/Windows) denominado *eTool Environment – Component Architecture* (eteC). El programa eteC forma parte del estándar KNX y proporciona un modelo de objetos (DOM: *Domain Object Model*) y una API para acceder a los recursos KNX. La arquitectura de ETS incluye los siguientes componentes (véase la Figura 5-7):

- Interfaz de usuario de ETS. Utilizada para configurar manualmente los dispositivos de la instalación.
- *Plug-ins.* Permiten añadir nuevas funciones de programadores externos al entorno.
- *Hawk*. Es un gestor basado en bases de datos que se utiliza para descargar el software a los dispositivos KNX.
- *Falcon*. Es una librería de interfaz con el modelo DCOM que permite diseñar aplicaciones Windows para acceder a la red KNX.
- *Eagle*. Es un componente similar a *Falcon*, pero destinado al acceso a la base de datos estándar de ETS (ETS DB), que contiene la información de los diferentes dispositivos KNX, incluyendo su descripción, programas de aplicación y funcionalidades. *Eagle* traslada la estructura física de la base de datos del repositorio ETS a un modelo abstracto de objetos del dominio KNX persistentes (DOM).

[1] Proyecto BASYS: http://www.basys2003.org

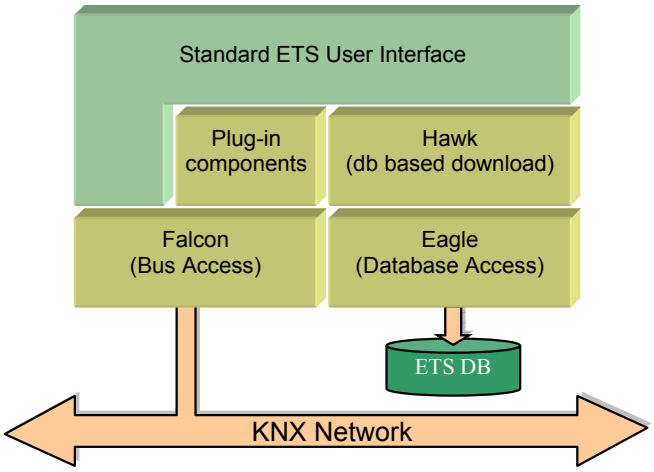


Figura 5-7. Arquitectura Software de ETS. Fuente: [KNX 04].

Para facilitar el acceso al Modelo de Objetos de Dominio (DOM) de eteC en la generación de código se utilizará el editor de macros del conjunto de herramientas *ITTools*[2], que se instala como *plugin* de ETS. Este editor permite la manipulación de todo el modelo de objetos de KNX con diferentes lenguajes (Java, Visual Basic, XML, etc.), y por tanto la creación completa de un proyecto KNX a partir de código (véase la Figura 5-8).

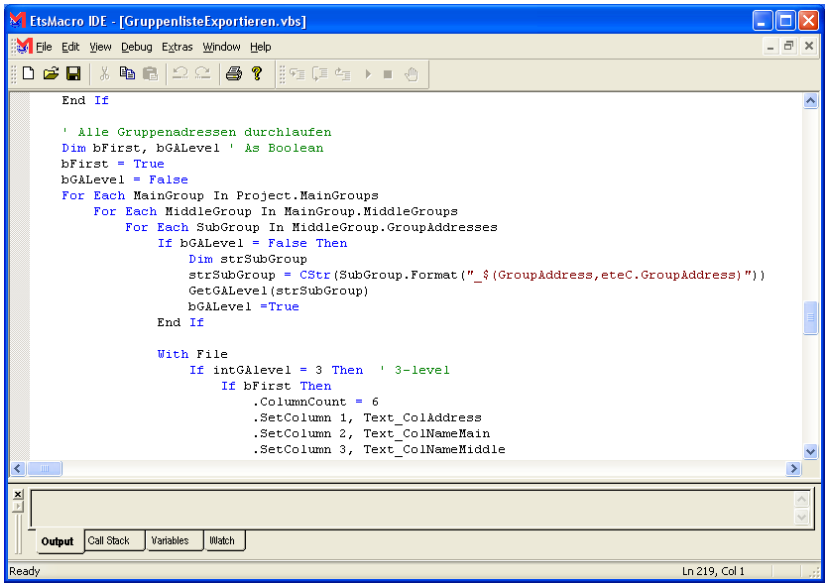


Figura 5-8. Editor de macros para ETS.

5.3.3 Un Metamodelo para KNX/EIB

La metodología expuesta para el desarrollo de sistemas domóticos está basada en el enfoque MDA, que propugna el empleo de modelos en todo el ciclo de desarrollo del producto. Esto implica la existencia de metamodelos en todos los niveles (CIM – PIM –

[2] http://www.it-gmbh.de/en/products/ittools.htm

PSM) para la creación de modelos conforme a ellos y la definición de transformaciones modelo a modelo y modelo a texto. Por lo tanto, debe existir un metamodelo para cada una de las plataformas utilizadas en el nivel PSM que permita definir las reglas de transformación entre el metamodelo de origen en el nivel PIM y cada uno de los de destino en el nivel PSM. B. M. Duc [Duc 07] incluye entre sus recomendaciones para lograr una metodología "a prueba de futuro" la necesidad de utilizar modelos en el nivel PSM, en vez de código fuente, y generar el código a partir de dichos modelos (transformaciones M2T). Si el modelo es bueno, perdurará en el tiempo y siempre será posible generar código en distintos lenguajes aunque éstos cambien o evolucionen.

En el caso particular de la plataforma KNX/EIB es posible extraer un metamodelo a partir del Modelo de Objetos de Dominio (DOM) de eteC, descrito en el apartado anterior. Existen tres modelos para este dominio: el de fabricantes (*Manufacturer*), el de la asociación de normalización (*EIBA*) y el de proyectos (*Project*). En la Figura 5-9 se puede observar el modelo de proyectos, donde aparecen los conceptos detallados con anterioridad en el apartado 5.3.1, como dispositivos, direcciones de grupo, direcciones físicas (en forma de áreas, líneas y dispositivos), canales, programas de aplicación, etc.

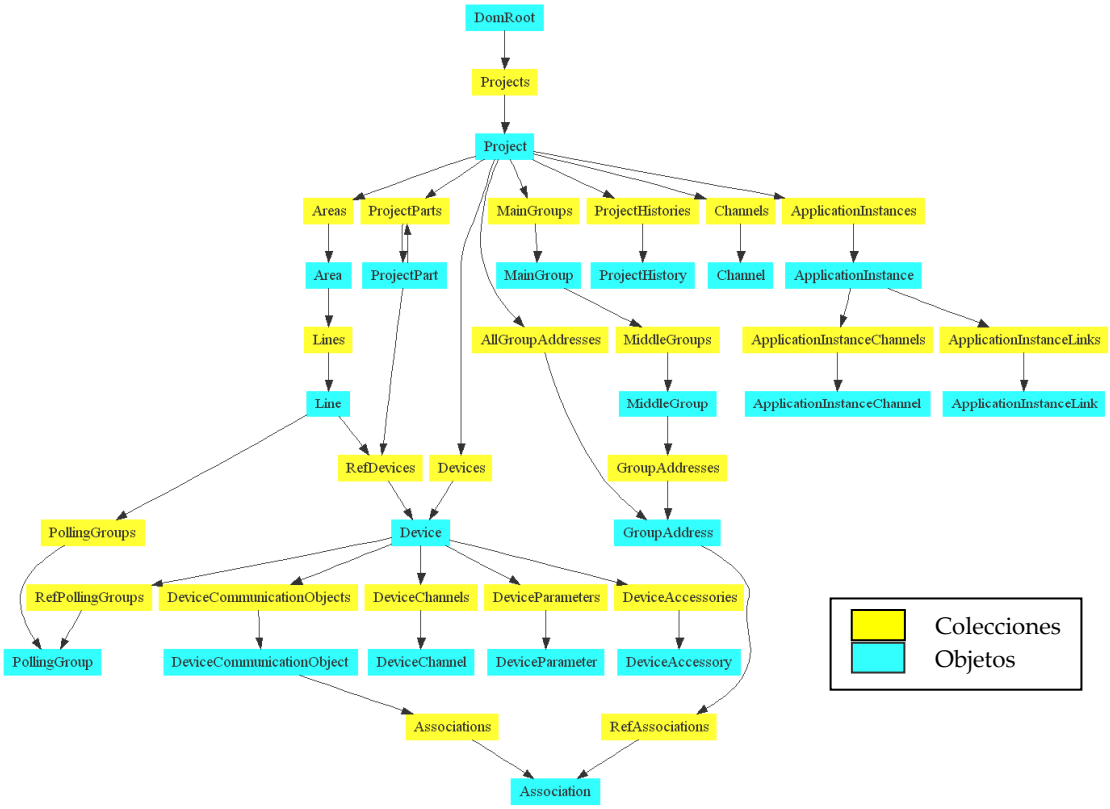


Figura 5-9. Modelo de Objetos del Domino para proyectos KNX/EIB. Fuente: documentación *ITTools (eteC DOM, Hawk documentation)*.

A partir de este modelo, del conocimiento y la experiencia con este sistema, se ha construido, como parte del trabajo de esta Tesis, un metamodelo de KNX/EIB empleando el entorno *Eclipse*, y en particular el *plugin* EMF (*Eclipse Modeling*

Framework) que da soporte al desarrollo con metodología MDA. Dicho metamodelo se presenta en la Figura 5-10. En él se han incluido las clases necesarias para la creación de un proyecto completo, simplificando a su vez al máximo la estructura del diagrama en su conjunto.

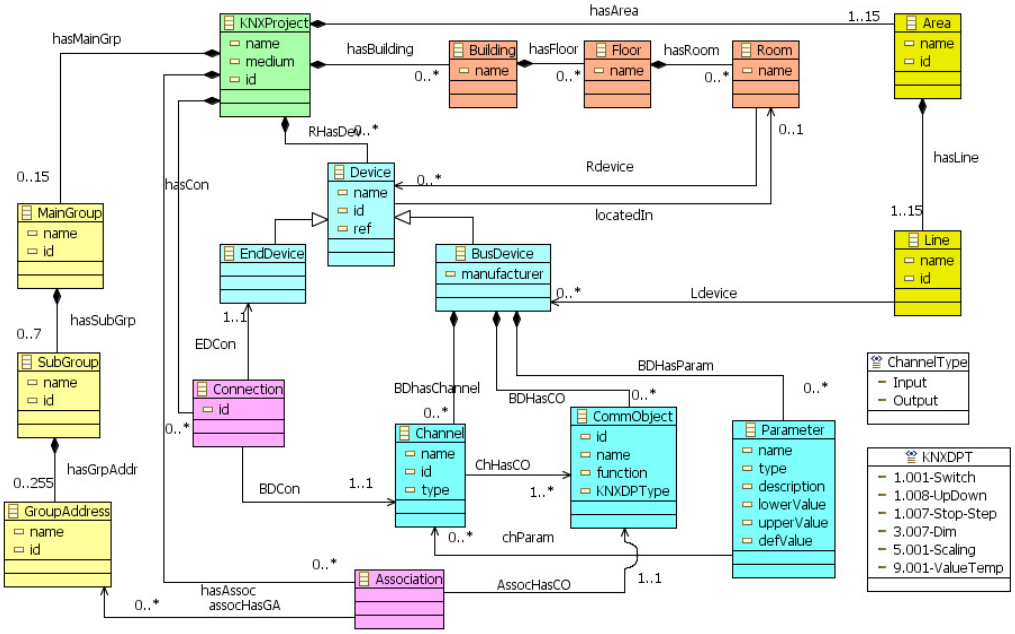


Figura 5-10. Metamodelo de nivel PSM para KNX/EIB.

La clase *KNXProject* actúa como raíz del metamodelo, a partir de la cual se crean las clases para la topología del bus (*Area, Line* y *BusDevice*), las correspondientes a las direcciones de grupo (*MainGroup*, *SubGroup* y *GroupAddress*) y las de distribución de dispositivos en estancias (*Building*, *Floor* y *Room*). La clase *Device* se especializa en dispositivos finales (*EndDevice*), que son elementos pasivos como bombillas, pulsadores o detectores, y dispositivos de bus (*BusDevice*) que son los propios del sistema KNX, e integran un microcontrolador con la pila del protocolo de comunicaciones y ejecutan un programa de aplicación para comunicarse con otros dispositivos del bus. Este último tipo de aparatos (*BusDevice*) incluye en su programa de aplicación los elementos descritos en el apartado 5.3.1: objetos de comunicación (*CommObject*), parámetros de configuración (*Parameter*) o canales (*Channel*) que representan las salidas o entradas (según el atributo *type*, que es de tipo *ChannelType*) a las que se pueden conectar elementos externos de tipo final (*EndDevice*). El enlace entre objetos de comunicación (*CommObject*) se realiza mediante la clase *Association*, que permite la asignación de direcciones de grupo a los objetos. Además, los objetos de comunicación tienen asignado un tipo de datos (atributo *KNXDPTType*) de los disponibles para la plataforma KNX (enumerado KNXDPT). Para simplificar tanto el metamodelo como las transformaciones se ha recogido un subconjunto de los tipos Konnex (DPTs o *Data Point Types*) más importantes.

En este apartado se han propuesto las claves para abordar con éxito la generación de aplicaciones para la plataforma KNX/EIB siguiendo la metodología dirigida por

modelos. En primer lugar, la importancia de la tecnología KNX/EIB justifica el esfuerzo realizado para utilizarla como una de las plataformas destino a nivel PSM. Por otra parte, se han descrito los conceptos fundamentales del dominio KNX/EIB y las características de la arquitectura software de las herramientas utilizadas para su diseño. Esto ha permitido crear un metamodelo para esta plataforma y definir una vía para la generación de código mediante plantillas en lenguajes de propósito general (como *JavaScript* o *VBScript*). Este aspecto es fundamental en el caso de KNX/EIB, ya que las librerías de programas de los fabricantes se distribuyen en paquetes codificados, que de otra manera no serían accesibles (o difícilmente accesibles realizando trabajos de ingeniería inversa).

Con esta base, en la siguiente sección se describen las transformaciones para llegar a modelos de la plataforma para obtener el código para configurar el sistema domótico.

5.4 Transformaciones CIM – PIM – PSM

Las transformaciones son una parte esencial para dar soporte al desarrollo dirigido por modelos ya que permiten convertir modelos entre diferentes metamodelos manteniéndolos sincronizados. Esto va a ser fundamental para completar el ciclo de desarrollo de aplicaciones en el entorno de la metodología propuesta, en la que se han definido metamodelos en los tres niveles planteados por MDA: CIM, PIM y PSM.

Czarnecki [Czarnecki 03] distingue dos tipos de transformación en función de la fase en que se realiza y del tipo de artefacto generado: transformaciones M2M (Modelo a Modelo) y M2T (Modelo a Texto). En la metodología propuesta en esta Tesis se realizan transformaciones M2M para hacer evolucionar los modelos creados con el DSL en el CIM hasta llegar a un modelo conforme al metamodelo para el PSM de destino. En este nivel PSM se realizarán transformaciones M2T para obtener el código ejecutable.

De acuerdo con los principios de MDA, las transformaciones entre modelos incluidas en cualquier proceso de desarrollo deben ser automatizadas, al menos en cierta medida, pero además, como paso previo a la automatización, dichas transformaciones deben ser formalizadas con el objetivo de reducir la posibilidad de introducir errores en los modelos que se van generando a lo largo del proceso de desarrollo.

Según [MDA 03], "la descripción de los transformaciones puede realizarse en lenguaje natural, un algoritmo en un *action language* o un modelo en un lenguaje de transformación". Actualmente el método más utilizado en la definición de reglas de transformación sigue siendo el lenguaje natural, aunque existe la posibilidad de formalizar las reglas de transformación entre los modelos propuestos por medio de gramáticas de grafos [Sprinkle 03], dado que resultan más intuitivas y proporcionan soporte directo para implementarlas. Las transformaciones basadas en grafos son una aproximación declarativa, visual y lo más importante, formal, a las transformaciones de modelos. En esencia, son un tipo de transformaciones basadas en reglas

representadas con diagramas. El hecho de que los modelos en general se puedan representar como grafos aumenta su atractivo desde el punto de vista del desarrollo software dirigido por modelos [Selic 03]. Evidentemente, este tipo de transformaciones resulta especialmente útil e intuitivo cuando se trata de definir transformaciones entre modelos definidos con lenguajes visuales, cuyo más claro exponente es UML.

La estrategia seguida en este trabajo de Tesis consiste en la utilización de una aproximación basada en grafos [Czarnecki 03][Tratt 05] para formalizar las transformaciones y, adicionalmente, dar un paso más en la dirección de la automatización de dichas transformaciones, dado que existen proyectos y herramientas que proporcionan la funcionalidad necesaria para automatizar transformaciones entre modelos expresados con grafos, como VIATRA2 [VIATRA 06], VMTS [Leven 04], AToM3 [Lara 02], GReAT [GreAT 08][Vizhanyo 04], MOFLON [MOFLON 08], Gmorph [Sendall 03b], MOTMOT [MoTMoT 06], AGG [AGG 08] y EMT [Biermann 06].

Para este trabajo se ha utilizado la herramienta de transformación de grafos AGG [AGG 08] del Instituto para Informática Teórica y Técnicas de *Software* de la Universidad Técnica de Berlín. AGG permite el prototipado rápido de aplicaciones con grafos estructurados complejos y dispone de un potente motor de transformación que puede ser utilizado con su propia interfaz (véase la Figura 5-11) o bien integrado en otras aplicaciones mediante una API. Por otra parte, la interfaz gráfica de AGG facilita la escritura de reglas así como la verificación de su aplicación a los modelos y la comprobación de forma interactiva. Además, existen proyectos como EMT [Biermann 06] que facilitan la integración de este motor de transformaciones de grafos en el entorno Eclipse utilizado en esta Tesis, lo que posibilita la automatización de las transformaciones definidas.

La expresión de transformaciones de modelos como gramáticas de grafos se basa en la definición de un conjunto de reglas. Estas reglas están formadas por una parte izquierda (LHS o *Left Hand Side*), una parte derecha (RHS o *Right Hand Side*) y, opcionalmente, por condiciones de aplicación negativa (NAC o *Negative Application Condition*). Estas reglas se aplicarán a un modelo de objetos enlazados en forma de grafo. En el caso particular de este trabajo, se trata de un modelo conforme a un metamodelo, que a su vez es conforme al metamodelo de clases de EMF.

En la parte izquierda (LHS) se define un (sub)grafo patrón que representa las precondiciones estructurales que deben cumplirse para la aplicación de la regla. En la parte derecha (RHS) se define un (sub)grafo de sustitución que describe el resultado o postcondición de la aplicación de la regla. Las condiciones de aplicación negativa (NAC) se definen de la misma forma que LHS y RHS, pero representan condiciones estructurales que no deben cumplirse para aplicar la regla. Cada vez que se encuentra una correspondencia con el patrón LHS en el modelo origen, se sustituye en el modelo destino por el grafo de sustitución RHS [Sendall 03]. Así, todos los objetos y relaciones presentes en LHS pero no en RHS se borran, y todos los objetos y enlaces presentes en RHS pero no en LHS se crean.

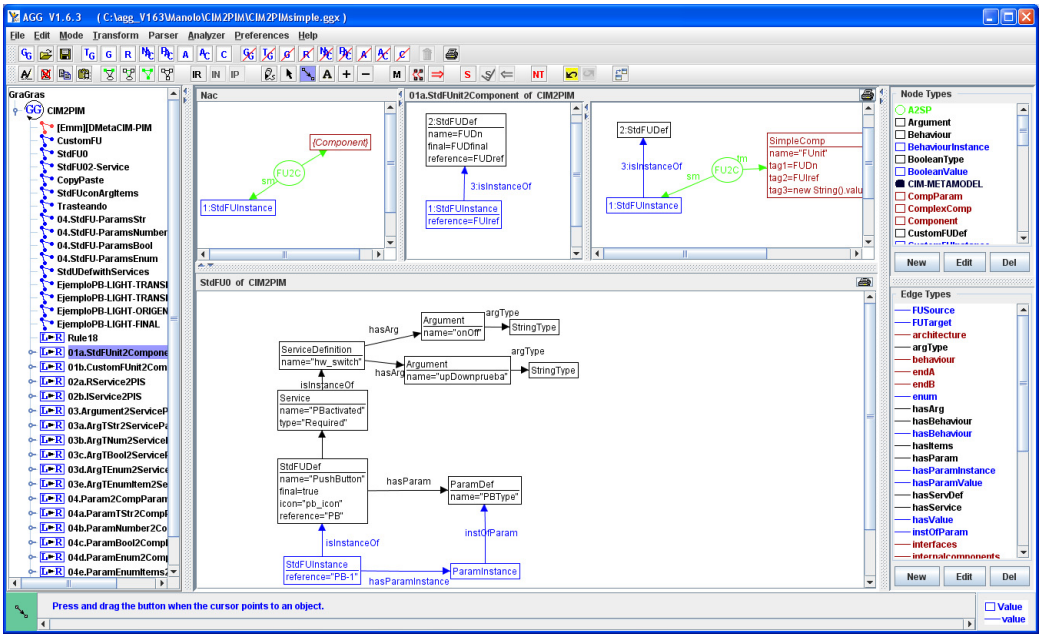


Figura 5-11. Herramienta AGG para transformación de grafos.

Para la especificación de las transformaciones es habitual establecer correspondencias entre los nodos de LHS con los de RHS y los de la(s) condiciones NAC. Además, tanto LHS como NAC pueden contener constantes o variables como valores de los atributos del nodo. Dichas variables pueden ser utilizadas en la parte derecha para declarar operaciones que asignen valores a los atributos de los nodos. La forma de establecer las correspondencias entre nodos de LHS, RHS y NAC varía según la herramienta utilizada, aunque lo más frecuente es utilizar números que identifican a los nodos y asociaciones, como es el caso de AGG.

A continuación se detallan las transformaciones entre los diferentes niveles MDA como soporte a l a generación automática de componentes a partir del modelo obtenido con el DSL.

5.4.1 Transformaciones CIM – PIM

La especificación de un sistema domótico de acuerdo con la metodología propuesta en la Figura 5-2 comienza con la captura de requisitos con el DSL que recoge sus características mediante conceptos del dominio y con un alto nivel de abstracción. El resultado será un modelo de nivel CIM de acuerdo con el metamodelo conforme al cual se ha creado dicho DSL. Este modelo será el origen de una transformación M2M a otro modelo del nivel PIM, que será conforme al metamodelo de componentes de V³Studio. En este apartado se presentan las transformaciones necesarias para afrontar con éxito este proceso, manteniendo la coherencia entre los modelos de ambos niveles.

En la Tabla 5-1 se resumen las reglas de transformación más importantes entre los metamodelos de nivel CIM y PIM en lenguaje natural, que servirán como punto de partida para su posterior definición de manera formal mediante gramática de grafos.

ID	Descripción de la regla de transformación
1.	Cada unidad funcional *FUnitInstance* del modelo CIM se transformará a un elemento *SimpleComponent* del modelo PIM que incluya la información contenida en *FUnitInstance* y en la definición de unidad funcional *FUnitDefinition* a la que está asociada.
1a.	Las unidades funcionales estándar *StdFUInstance* del CIM se transformarán a *SimpleComponent* del modelo PIM sin máquina de estado asociada.
1b.	Las unidades funcionales personalizadas *CustomFUInstance* con comportamiento *Behaviour* asociado del CIM se transformarán a *SimpleComponent* del modelo PIM con máquina de estado *StateMachine* asociada.
2.	Los servicios de las definiciones de unidades funcionales que se encuentre asociados a instancias de unidades funcionales del modelo CIM se transformarán a Puerto – Interfaz – Servicio (*Port – Interface – Service*) del modelo PIM asociados al componente simple creado a partir de la unidad funcional.
2a.	Si el servicio del modelo CIM es requerido, la asociación puerto – interfaz del modelo PIM será requerida (*requiredInterface*).
2b.	Si el servicio del modelo CIM es implementado, la asociación puerto – interfaz del modelo PIM será provista (*providedInterface*).
3.	Los argumentos de los servicios del catálogo de unidades funcionales que estén instanciadas por *FUnitInstance* en el CIM se transformarán en *ServiceParam* del PIM, asociados a los correspondientes componentes simples.
3a.	Si el argumento está asociado a un tipo *StringType*, el *ServiceParam* del PIM tendrá un literal "*String*" en su atributo *type*.
3b.	Si el argumento está asociado a un tipo *NumberType*, el *ServiceParam* del PIM tendrá un literal "*Number*" en su atributo *type*.
3c.	Si el argumento está asociado a un tipo *BooleanType*, el *ServiceParam* del PIM tendrá un literal "*Boolean*" en su atributo *type*.
3d.	Si el argumento está asociado a un tipo *EnumType*, el *ServiceParam* del PIM tendrá un literal "*Enum*" en su atributo *type*.
3e.	Cada *Item* asociado al *EnumType* añadirá un valor igual al contenido en su atributo *name* al atributo *type* del argumento de destino en el modelo del PIM.
4.	Cada parámetro de unidad funcional, asociado a una definición de unidad funcional instanciada por un *ParamInstance* asociado a una *StdFUInstance* del modelo CIM se transformará en un parámetro de componente *CompParam* del modelo PIM.
4a.	Si el argumento está asociado a un tipo *StringType*, el *CompParam* del PIM tendrá un literal "*String*" en su atributo *type*.
4b.	Si el argumento está asociado a un tipo *NumberType*, el *CompParam* del PIM tendrá un literal "*Number*" en su atributo *type*.
4c.	Si el argumento está asociado a un tipo *BooleanType*, el *CompParam* del PIM tendrá un literal "*Boolean*" en su atributo *type*.
4d.	Si el argumento está asociado a un tipo *EnumType*, el *CompParam* del PIM tendrá un literal "*Enum*" en su atributo *type*.
4e.	Cada *Item* asociado al *EnumType* añadirá un valor igual al contenido en su atributo *name* al atributo *type* del parámetro de destino en el modelo del PIM.
4f.	El nombre del *Item* seleccionado de la colección definida en *EnumType* del modelo CIM se asignará al atributo *value* del *CompParam* del modelo del PIM.
5.	Cada enlace entre unidades funcionales *FULink* del modelo CIM se transformará en un enlace de tipo *PortLink* entre los componentes y servicios de destino en el modelo PIM.

Tabla 5-1. Reglas de transformación CIM-PIM expresadas en lenguaje natural.

Para la definición de transformaciones exógenas con gramática de grafos como las que aquí se proponen (entre modelos origen y destino conforme a diferentes metamodelos), es necesario establecer un metamodelo de referencia que sirve como estructura de apoyo para las diferentes transformaciones. Esto significa que las transformaciones en sí tienen su propio metamodelo con el que son conformes. La representación de este metamodelo mediante un grafo resulta difícil de interpretar, ya que incluye todas las clases de los metamodelos de los niveles CIM y PIM, así como las clases y asociaciones de transformación. Por lo tanto, para facilitar su comprensión se ha optado por mostrarlo por partes en las propias reglas de transformación mediante clases representadas con círculos de color verde con asociaciones con las clases del metamodelo CIM de origen (asociaciones sm: *source model*) y el metamodelo PIM de destino (asociaciones tm: *target model*). Desde la Figura 5-12 a la Figura 5-29 pueden observarse las reglas de transformación basadas en grafos para las transformaciones automatizables entre modelos del nivel CIM y sus correspondientes elementos del nivel PIM de la Tabla 5-1. Las clases de la parte del catálogo del modelo PIM se representan en color negro, y las correspondientes a la aplicación en color azul. En color rojo se muestran las clases del modelo de destino (nivel PIM). Estos mismos colores se han utilizado al hacer referencia a las clases implicadas en la regla de transformación en el texto que las describe. A continuación se describen las reglas en detalle manteniendo la numeración utilizada en la Tabla 5-1.

1. ***FUnit2Component***. Por cada objeto instancia de unidad funcional *FUnitInstance* asociada a una unidad funcional *FUnitDefinition* del modelo CIM se creará un componente simple *SimpleComp* del modelo PIM. Puesto que esta clase se especializa en las clases *StdFUInstance* y *CustomFUInstance*, se definen dos reglas:

1a. ***StdFUnit2Component*** (Figura 5-12). Por cada objeto *StdFUInstance* asociado a una unidad funcional estándar *StdFUDef* del modelo CIM se creará un componente simple *SimpleComp* del modelo PIM con el nombre *"FUnit"*, y en sus etiquetas se almacenará el nombre de la *StdFUDef*, la referencia de la instancia *StdFUInstance* y el atributo *final* de la definición de Unidad Funcional (mediante la operación Java *new String().valueOf(FUDfinal)*). Para ello se utilizan las variables *FUDn, FUIref* y *FUDfinal*, declaradas en la LHS de la regla y utilizadas en la RHS. La condición NAC evita que se vuelva a aplicar la regla a instancias de unidades funcionales ya transformadas a componentes. El nodo *FU2C* representa la clase del metamodelo de referencia que establece correspondencia de transformación desde una clase *StdFUInstance* del CIM a la clase *SimpleComp* del PIM.

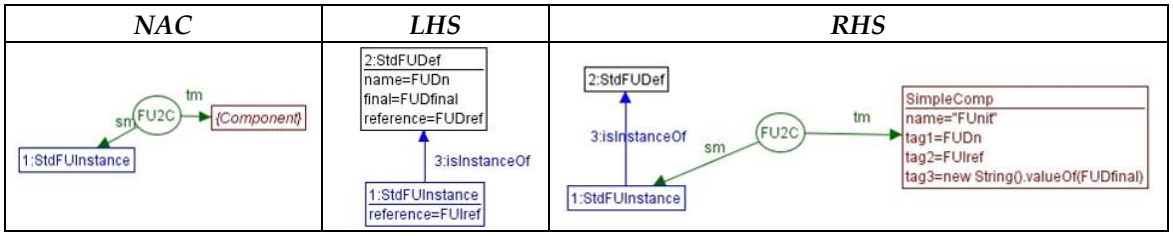


Figura 5-12. Regla de transformación *StdFUnit2Component*.

1b. ***CustomFUnit2Component*** (Figura 5-13). Por cada objeto *CustomFUInstance* asociado a una unidad funcional *CustomFUDef* del modelo CIM se creará un componente simple *SimpleComp* del modelo PIM con el nombre "FUnit", y en sus etiquetas se almacenará el nombre de la *CustomFUDef*, la referencia del objeto *CustomFUInstance* y el atributo *final* de la definición de Unidad Funcional (mediante la operación Java *new String().valueOf(FUDfinal)*). Para el objeto *BehaviourInstance* asociado al comportamiento de la definición de unidad funcional *Behaviour* se creará un objeto de tipo *StateMachine* cuyo nombre se expresará en el formato <*BehaviourInstance.name*>:<*Behaviour.name*>. Para ello se utilizan las variables *FUDn*, *FUIref*, *FUDfinal*, *Bn y BIn* declaradas en la LHS de la regla y utilizadas en la RHS. La condición NAC evita que se vuelva a aplicar la regla a instancias de unidades funcionales ya transformadas a componentes.

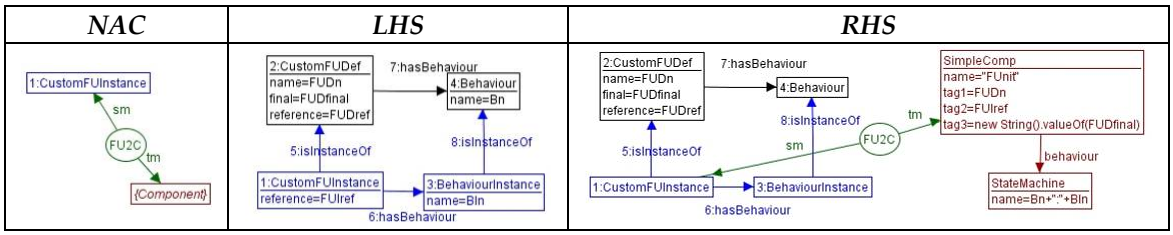


Figura 5-13. Regla de transformación *CustomFUnit2Component*.

2. ***Service2PIS.*** Por cada servicio *Service* de una unidad definición de unidad funcional *StdFUDef* que se encuentre asociado a una *StdFUInstance* se crearán los objetos *Port*, *Interface* y *Service* del modelo PIM asociados al *SimpleComp* creado a partir de la *StdFUInstance*. En el nombre del puerto se almacenará el nombre de la definición del servicio. En el nombre de la interfaz se almacenará el nombre del servicio, y en nombre del servicio del PIM los nombres del servicio y definición del servicio en el formato <*ServiceDefinition.name*>:<*Service.name*>. Puesto que los servicios pueden ser de dos tipos (*Required* o *Implemented*) se definen dos reglas, ya que en el metamodelo destino las interfaces se definen como *Required* o *Provided* mediante asociaciones diferentes, y no como un atributo de la clase:

2a. ***RService2PIS*** (Figura 5-14). Si el servicio del nivel CIM es de tipo *Required* la asociación entre los objetos *Port* e *Interfaz* creados en el PIM será de tipo *requiredInterface*. La condición NAC evita que se vuelva a aplicar la regla a servicios de definiciones de unidades funcionales ya transformadas a componentes.

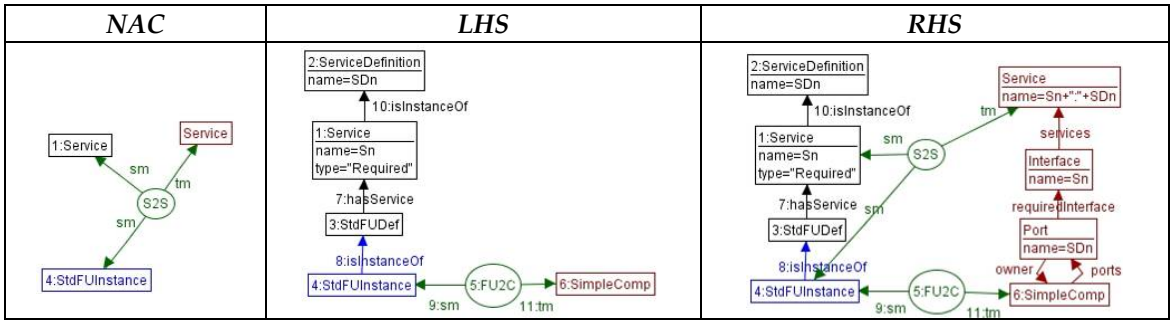


Figura 5-14. Regla de transformación *RService2PIS*.

2b. ***IService2PIS*** (Figura 5-15). Si el servicio del CIM es de tipo *Implemented* la asociación entre los objetos *Port* e *Interfaz* creados en el PIM será de tipo *providedInterface*. La condición NAC evita que se vuelva a aplicar la regla a servicios de definiciones de unidades funcionales instanciadas ya transformadas a componentes.

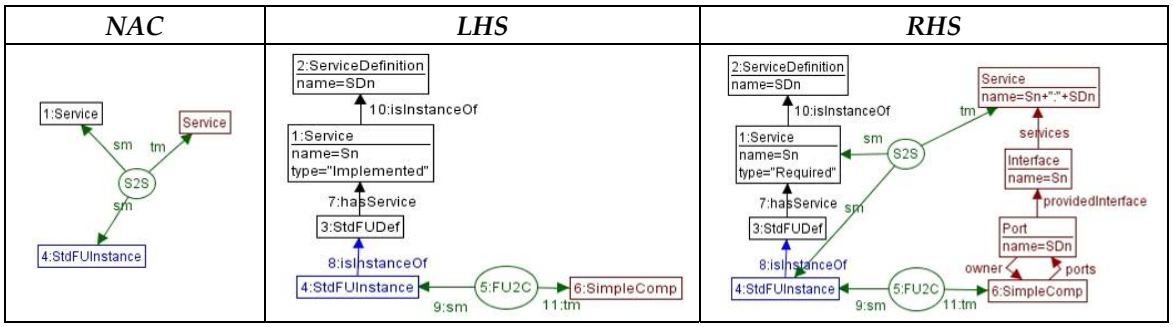


Figura 5-15. Regla de transformación *IService2PIS*.

3. ***Argument2ServiceParam*** (Figura 5-16). Por cada argumento *Argument* de la definición de servicio *ServiceDefinition* asociado a un servicio *Service* en el modelo CIM se creará un *ServiceParam* asociado al servicio de nivel PIM, con el mismo nombre que el argumento *Argument*. El tipo *type* del servicio dependerá del tipo del argumento de nivel CIM, y se asignará en las siguientes reglas (3a-3e). La condición NAC evita que se vuelva a aplicar la regla a argumentos de servicios ya transformados.

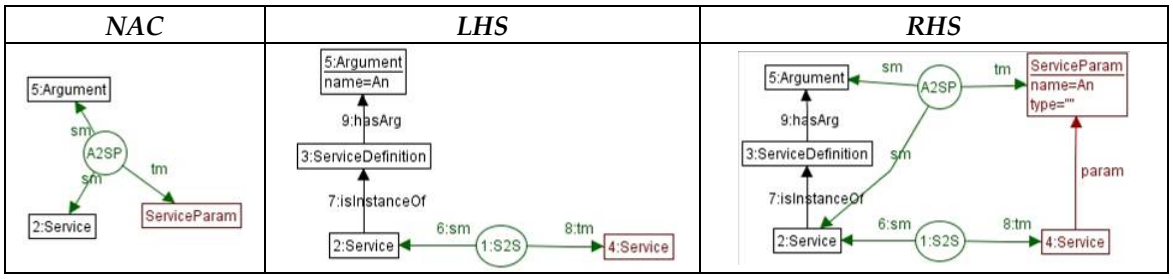


Figura 5-16. Regla de transformación *Argument2ServiceParam*.

3a. ***ArgTStr2ServiceParam*** (Figura 5-17). Si el argumento *Argument* tiene asociado un tipo *StringType* en el modelo de nivel CIM, el atributo *type* de *ServiceParam* de nivel PIM tomará el valor "*String*". La condición NAC evita que se vuelva a aplicar la regla a tipos de argumentos de servicios ya transformados.

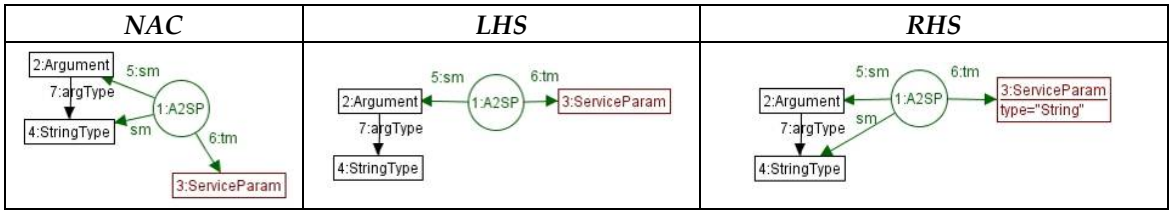


Figura 5-17. Regla de transformación *ArgTStr2ServiceParam*.

3b. ***ArgTNum2ServiceParam*** (Figura 5-18). Si el argumento *Argument* tiene asociado un tipo *NumberType* en el modelo de nivel CIM, el atributo *type* de *ServiceParam* de nivel PIM tomará el valor "*Number*". La condición NAC evita que se vuelva a aplicar la regla a tipos de argumentos de servicios ya transformados.

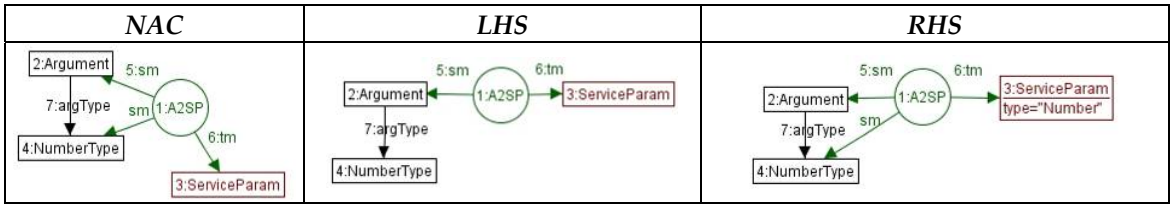


Figura 5-18. Regla de transformación *ArgTNum2ServiceParam*.

3c. ***ArgTBool2ServiceParam*** (Figura 5-19). Si el argumento *Argument* tiene asociado un tipo *BooleanType* en el modelo de nivel CIM, el atributo *type* de *ServiceParam* de nivel PIM tomará el valor "*Boolean*". La condición NAC evita que se vuelva a aplicar la regla a tipos de argumentos de servicios ya transformados.

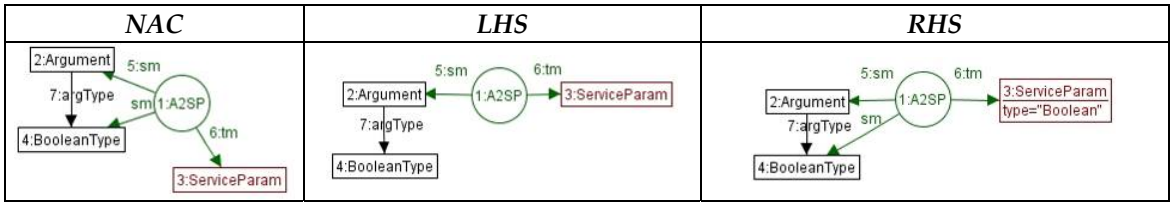


Figura 5-19. Regla de transformación *ArgTBool2ServiceParam*.

3d. ***ArgTEnum2ServiceParam*** (Figura 5-20). Si el argumento *Argument* tiene asociado un tipo *EnumType* en el modelo de nivel CIM, el atributo *type* de *ServiceParam* de nivel PIM tomará el valor "*Enum:*". La regla 3e asignará la lista de valores (*Items*) del enumerado al atributo *type*. La condición NAC evita que se vuelva a aplicar la regla a tipos de argumentos de servicios ya transformados.

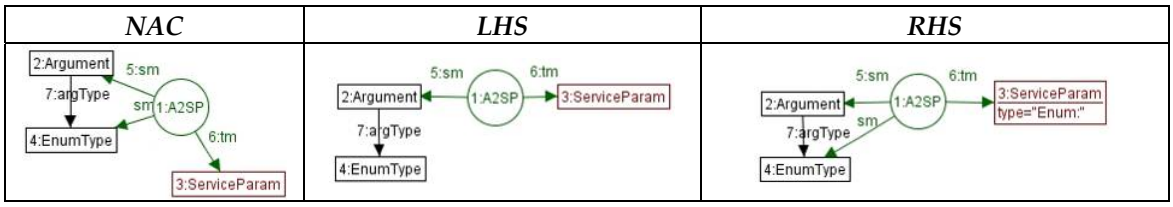


Figura 5-20. Regla de transformación *ArgTEnum2ServiceParam*.

3e. ***ArgTEnumItem2ServiceParam*** (Figura 5-21). Por cada *Item* asociado a un *EnumType* a su vez asociado a un argumento *Argument* en el modelo de nivel CIM, se añadirá al atributo *type* de *ServiceParam* de nivel PIM el nombre de dicho *Item* separado por comas. La condición NAC evita que se vuelva a aplicar la regla a *Items* ya transformados.

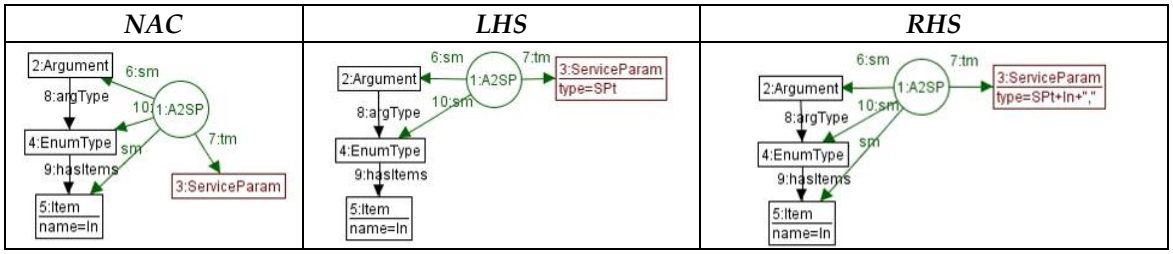


Figura 5-21. Regla de transformación *ArgTEnumItem2ServiceParam*.

4. ***Param2CompParam*** (Figura 5-22). Por cada definición de parámetro *ParamDef* asociado a una *StdFUDef*, que a su vez esté asociado a una instancia de parámetro *ParamInstance* asociada a una *StdFUInstance* del nivel CIM, se creará un parámetro de componente *CompParam* asociado al componente simple *SimpleComp* que se creó a partir del par *ParamInstance* - *StdFUInstance* mediante la regla 01a. El *CompParam* tendrá por nombre el de *ParamDef*. El tipo y valor se asignarán mediante las reglas 4a-4f.

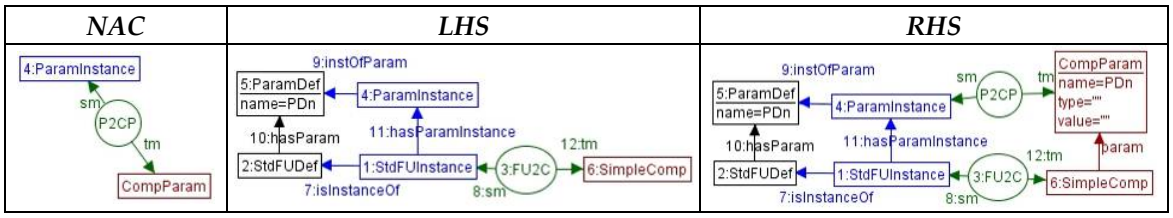


Figura 5-22. Regla de transformación *Param2CompParam*.

4a. ***ParamTStr2CompParam*** (Figura 5-23). Si el parámetro *ParamDef* tiene asociado un *StringType*, se asignará el literal *"String"* al componente *CompParam*, así como el valor del atributo *value* del *StringValue* asociado a la instancia del parámetro *ParamInstance* (que se almacena en la variable PSV). La condición NAC evita que se vuelva a aplicar la regla a tipos y valores de parámetros ya transformados.

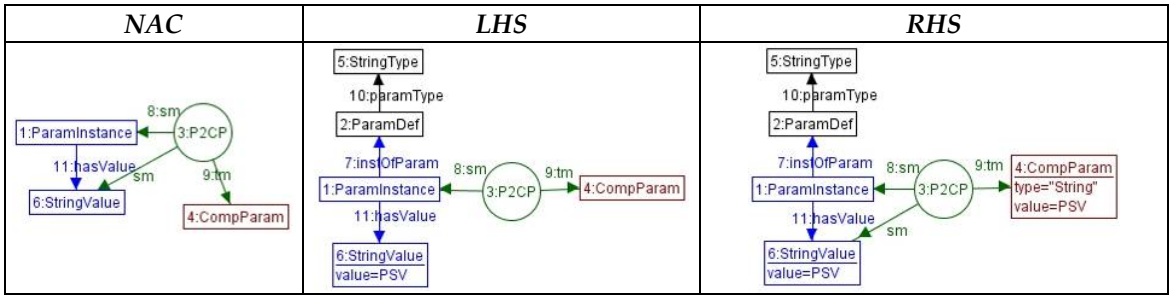


Figura 5-23. Regla de transformación *ParamTStr2CompParam*.

4b. ***ParamNumber2CompParam*** (Figura 5-24). Si el parámetro *ParamDef* tiene asociado un *NumberType*, se asignará el literal *"Number"* al componente *CompParam*, así como el valor del atributo *value* del *NumberValue* asociado a la instancia del parámetro *ParamInstance* (que se almacena en la variable PNV y se asigna mediante la operación Java *new String().valueOf(PNV)*). La condición NAC evita que se vuelva a aplicar la regla a tipos y valores de parámetros ya transformados.

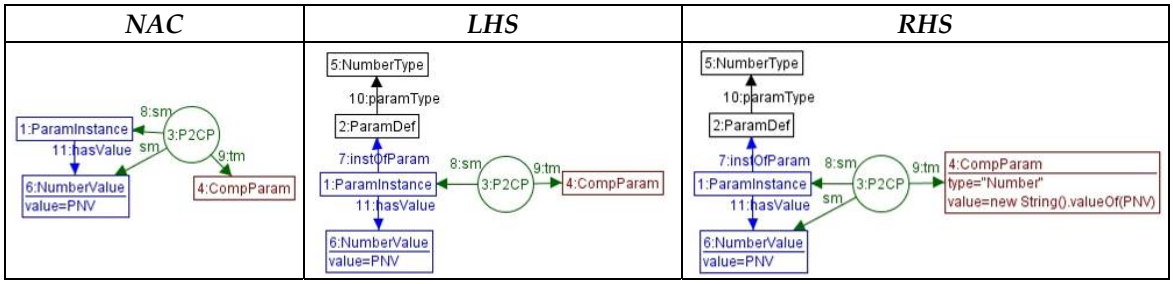


Figura 5-24. Regla de transformación *ParamNumber2CompParam*.

4c. ***ParamBool2CompParam*** (Figura 5-25). Si el parámetro *ParamDef* tiene asociado un *BooleanType*, se asignará el literal *"Boolean"* al componente *CompParam*, así como el valor del atributo *value* del *BooleanValue* asociado a la instancia del parámetro *ParamInstance* (que se almacena en la variable PNV y se asigna mediante la operación Java *new String().valueOf(PBV)*). La condición NAC evita que se vuelva a aplicar la regla a tipos y valores de parámetros ya transformados.

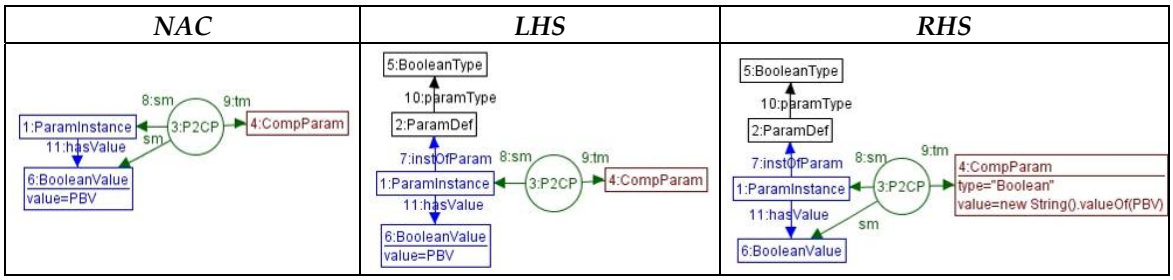


Figura 5-25. Regla de transformación *ParamBool2CompParam*.

4d. ***ParamEnum2CompParam*** (Figura 5-26). Si el parámetro *ParamDef* tiene asociado un *EnumType*, se asignará el literal *"Enum:"* al componente *CompParam*. La regla 4e asignará la lista de valores (*Items*) del enumerado al atributo *type*. La condición NAC evita que se vuelva a aplicar la regla a tipos y valores de parámetros ya transformados.

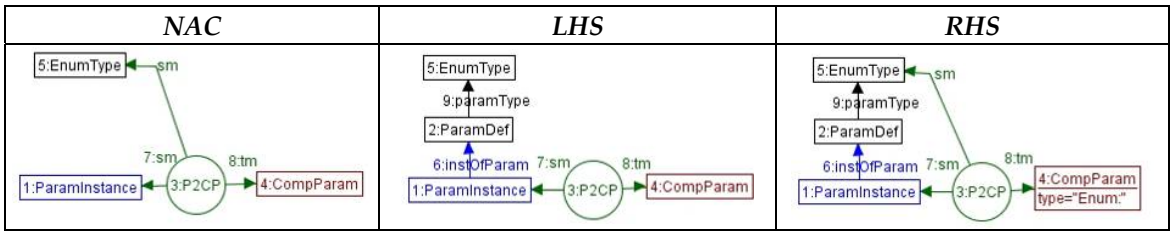


Figura 5-26. Regla de transformación *ParamEnum2CompParam*.

4e. ***ParamEnumItems2CompParam*** (Figura 5-27). Por cada *Item* asociado a un *EnumType* a su vez asociado a un parámetro *ParamDef* en el modelo de nivel CIM asociado a un *ParamInstance*, se añadirá al atributo *type* de *CompParam* de nivel PIM el nombre de dicho *Item* separado por comas. La condición NAC evita que se vuelva a aplicar la regla a *Items* ya transformados.

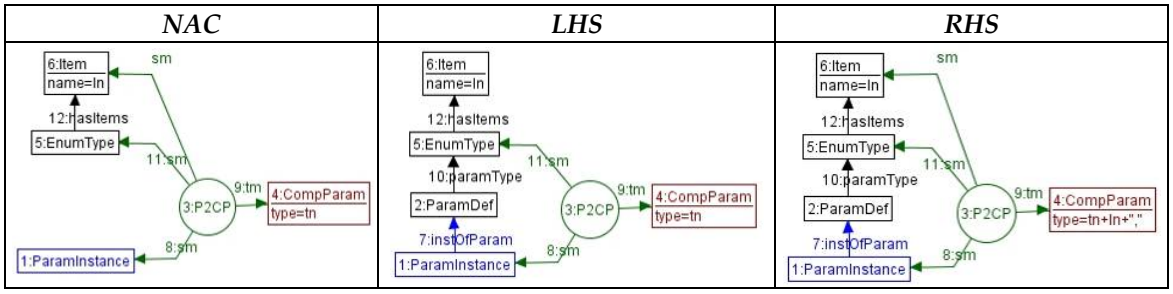


Figura 5-27. Regla de transformación *ParamEnumItems2CompParam*.

4f. ***ParamEnumSelItem2CompParam*** (Figura 5-28). Asigna al argumento *value* de *CompParam* en el nivel PIM el nombre del *Item* seleccionado de una colección de *Items* (de un *EnumType*). Esta asignación se hace a partir de la asociación de *ParamInstance* a la clase *SelectedItem*, que a su vez se encuentra asociado a un *Item* de un *EnumType*. dicho valor se almacena en la variable *In*. La condición NAC evita que se vuelva a aplicar la regla a un par *Item* - *SelectedItem*.

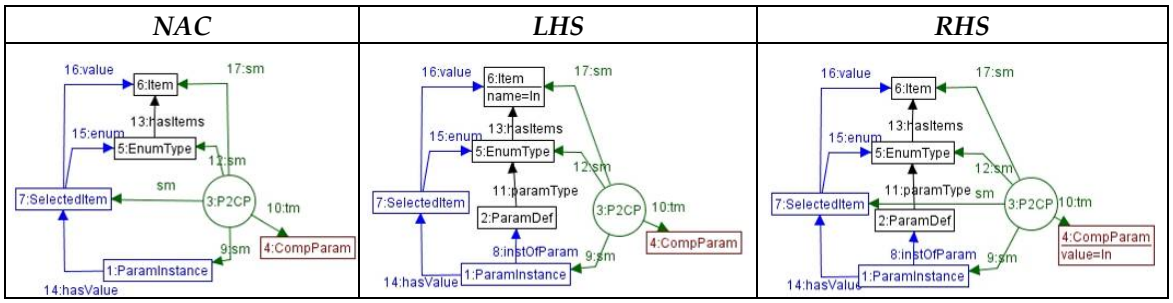


Figura 5-28. Regla de transformación *ParamEnumSelItem2CompParam*.

5. ***FUnitLink2PortLink*** (Figura 5-29). Por cada enlace *FULink* entre objetos *StdFUInstance* y sus correspondientes servicios *Service* de nivel CIM se creará en el PIM un enlace de tipo *PortLink* entre los puertos *Port* que se crearon en las reglas 2a y 2b, que a su vez se encuentran asociados a los *SimpleComp* creados a partir de las *StdFUInstance*. La condición NAC evita que se vuelva a aplicar la regla a enlaces *FULink* ya transformados.

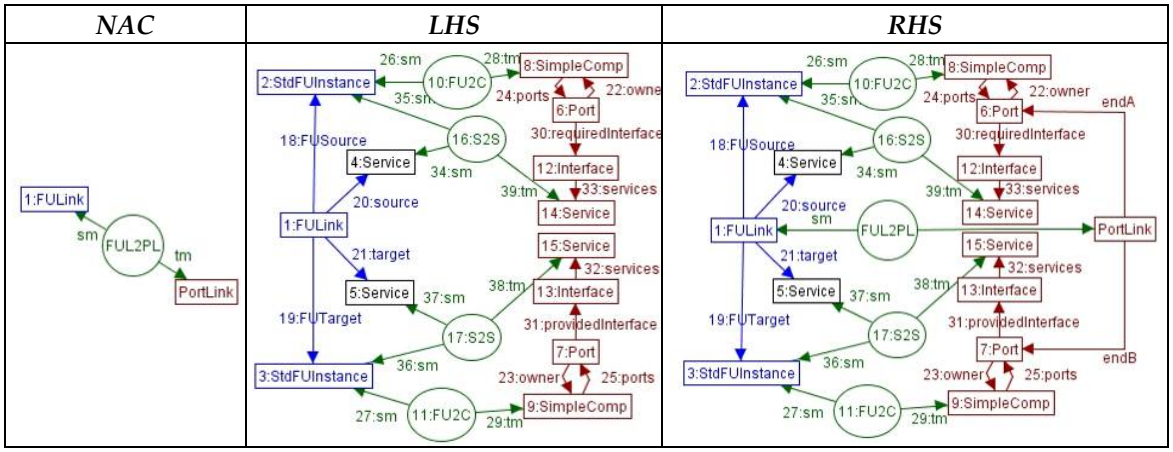


Figura 5-29. Regla de transformación *FUnitLink2PortLink*.

Para demostrar el correcto funcionamiento de las reglas, en la Figura 5-30 se presenta un ejemplo sencillo en el que enlazan cuatro unidades funcionales: el pulsador PB-1, que es una unidad funcional de tipo final, un elemento de entrada de conmutación SWI-1, uno de salida SWO-1 y una bombilla LO-1, también de tipo final. En la parte superior se muestran las unidades funcionales tal como aparecen en el DSL desarrollado para aplicaciones domóticas, y en la parte inferior el despliegue de objetos (instancias de clases del metamodelo origen de nivel CIM) correspondientes al diagrama representado con el DSL. Dada la complejidad del metamodelo instanciado, resulta muy difícil representar sobre el papel ejemplos más complejos, ya que su interpretación sería muy compleja. No obstante, todo este proceso de transformación es transparente al usuario. El modelo PIM de salida será la entrada para las transformaciones PIM – PSM, de forma que al final de todo el proceso se generará el código apropiado para la plataforma destino.

La Figura 5-31 muestra el modelo origen de nivel CIM y el modelo de nivel PIM obtenido como consecuencia de aplicar las reglas de transformación, así como la estructura de objetos de referencia utilizados para aplicar las transformaciones (círculos en color verde).

En el Anexo B se detalla la aplicación de las reglas, paso a paso, para la generación y refinamiento del modelo PIM a partir del modelo CIM de este ejemplo.

En los siguientes apartados se describe el proceso para realizar las transformaciones entre los modelos de nivel CIM y la plataforma KNX/EIB de nivel PSM, descrita en el apartado 5.3.

.

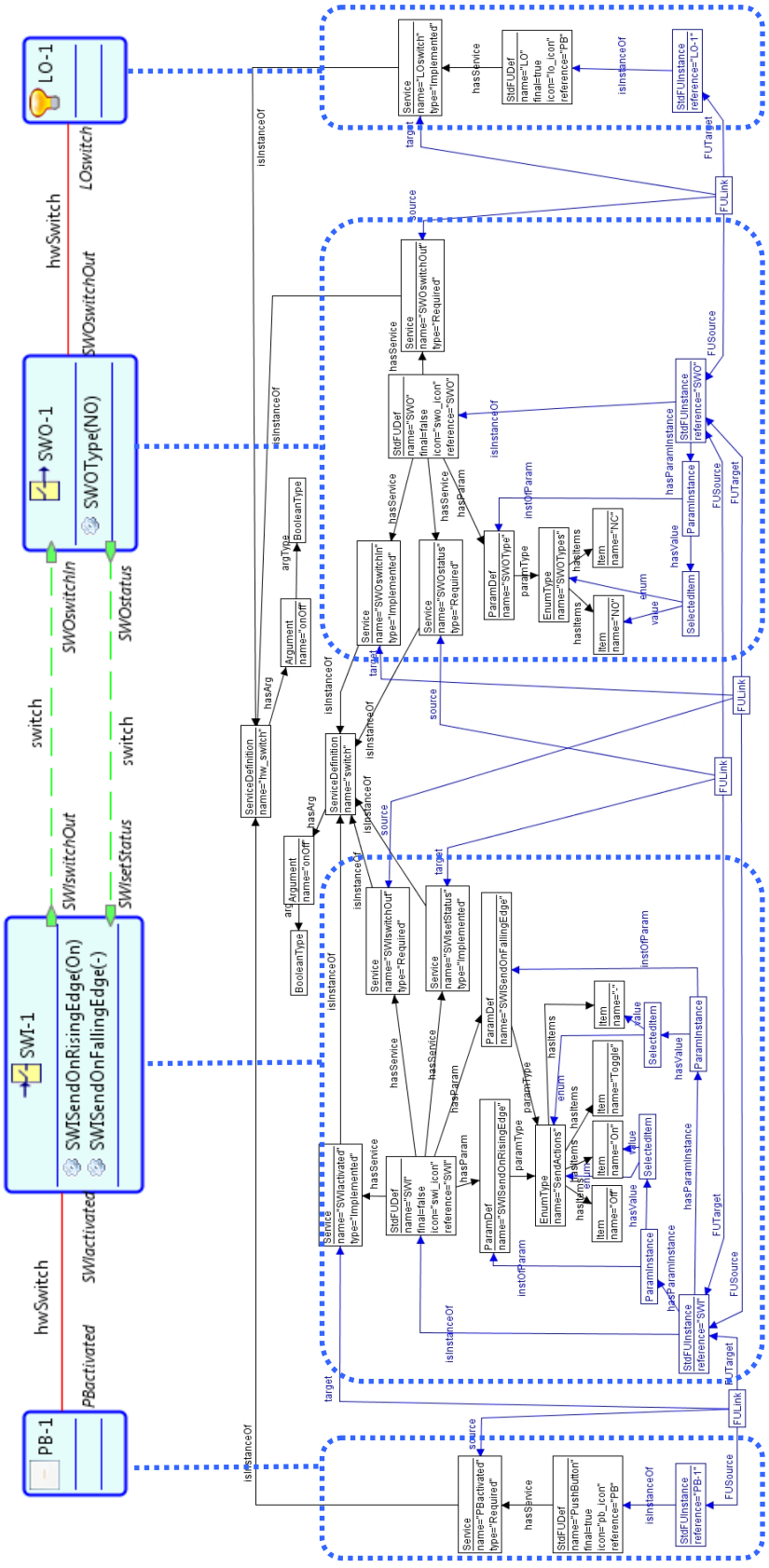


Figura 5-30. Ejemplo de aplicación de las transformaciones CIM - PIM. DSL y modelo CIM.

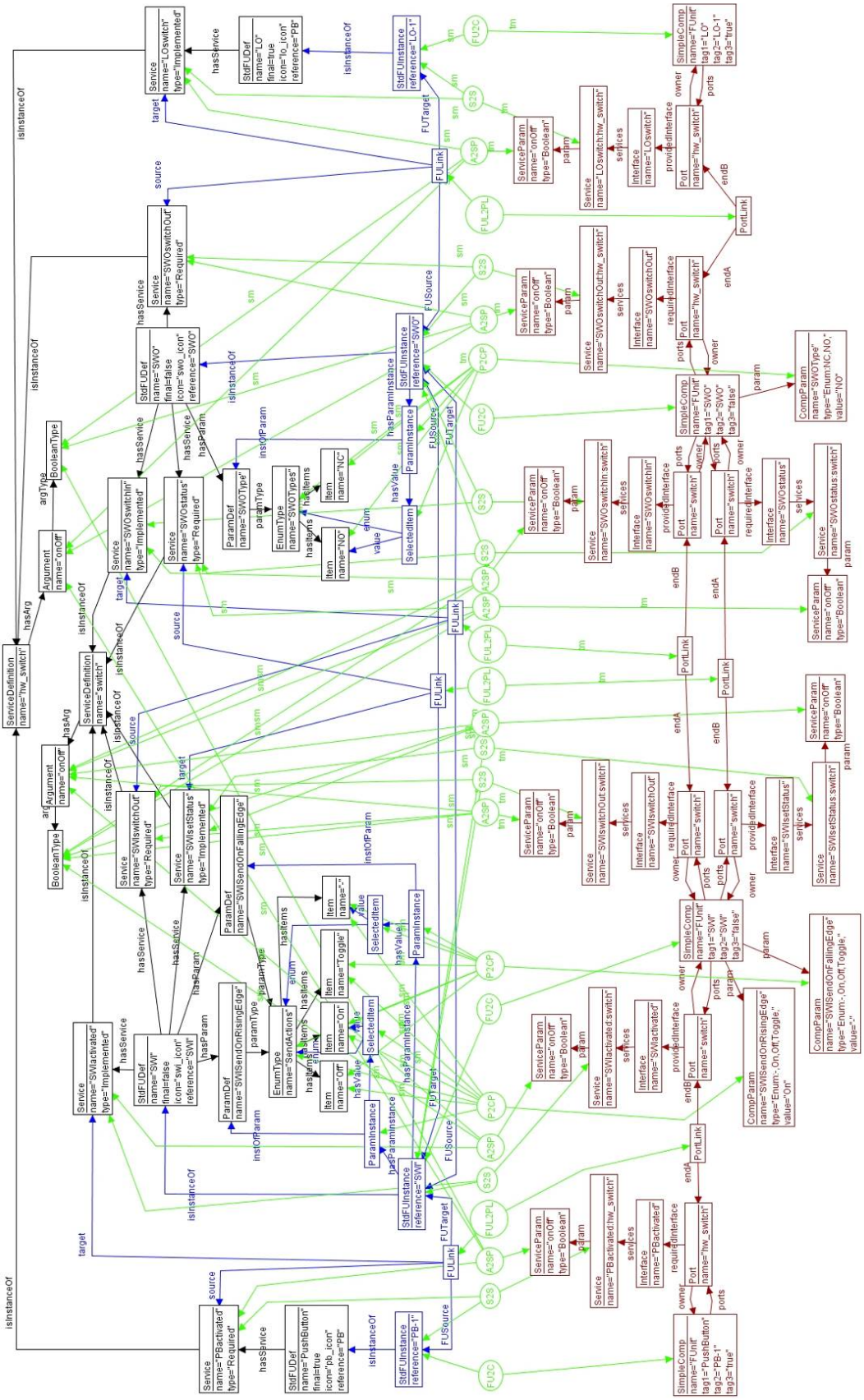


Figura 5-31. Ejemplo de aplicación de las transformaciones CIM - PIM. Modelos CIM y PIM.

5.4.2 Transformaciones PIM – PSM para la Plataforma KNX/EIB

De acuerdo con la metodología dirigida por modelos que se propone (véase la Figura 5-2), el proceso de desarrollo de un sistema domótico finaliza con la elección de una plataforma de ejecución y la generación de código ejecutable para la misma. Para ello es necesario definir dos tipos de transformaciones: una transformación M2M desde el modelo del nivel PIM intermedio a un modelo del nivel PSM seleccionado, y una segunda transformación M2T que obtenga el código a partir de dicho modelo.

Una de las principales ventajas del enfoque MDA radica en que a partir de un mismo modelo del nivel independiente de plataforma (PIM) se pueden obtener modelos y código para distintas plataformas del nivel PSM. En este trabajo de Tesis se describe el enfoque y las transformaciones necesarias para la plataforma KNX/EIB y se propone como trabajo futuro el desarrollo de las transformaciones para otras plataformas como Lonworks empleando la metodología propuesta.

En los niveles CIM y PIM se trabaja con unidades funcionales básicas para elevar el nivel de abstracción y conseguir la independencia de la plataforma. En el nivel PSM se necesita una configuración que permita agrupar dichas unidades funcionales en dispositivos domóticos comerciales, ya que éstos suelen incluir múltiples funciones (varias entradas, varias salidas, salidas combinadas con entradas, funciones lógicas, retardos, etc.). Esta configuración se realiza durante la transformación PIM – PSM tal como se presenta en la Figura 5-32. El modelo de nivel PIM se combina con el modelo de configuración, que debe crear un experto en la tecnología, y que establece las correspondencias de cada dispositivo del nivel PSM con los componentes derivados de las unidades funcionales del CIM. El metamodelo que da soporte a dicha configuración se muestra en la Figura 5-33. Este metamodelo permite seleccionar un componente e indicar a qué dispositivo estará asociado. Los dispositivos (*Device*) disponen de una serie de ranuras (*Slot*) donde se pueden ubicar los componentes de nivel PIM. La compatibilidad entre los *slots* y los componentes se determina comparando los atributos del componente con los del *Slot*.

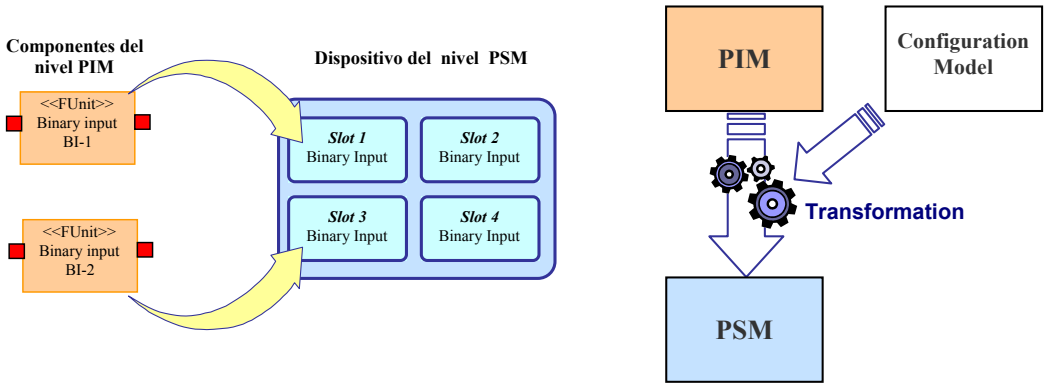


Figura 5-32. Mecanismo de transformación PIM – PSM y modelo de configuración.

En cada plataforma los dispositivos deberán estar catalogados empleando esta estructura predeterminada de modo que para realizar la configuración se pueda comprobar si un componente es compatible o no con el *slot* del dispositivo. Para ello se ha añadido la clase *Slot* al metamodelo propuesto en la Figura 5-10 para la plataforma KNX/EIB. Dicho metamodelo modificado puede observarse en la Figura 5-34.

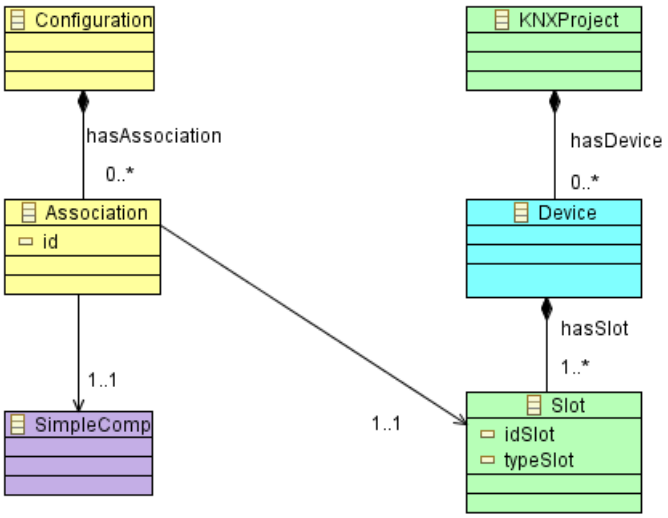


Figura 5-33. Metamodelo de configuración para las transformaciones PIM (parte izquierda) a PSM (parte derecha).

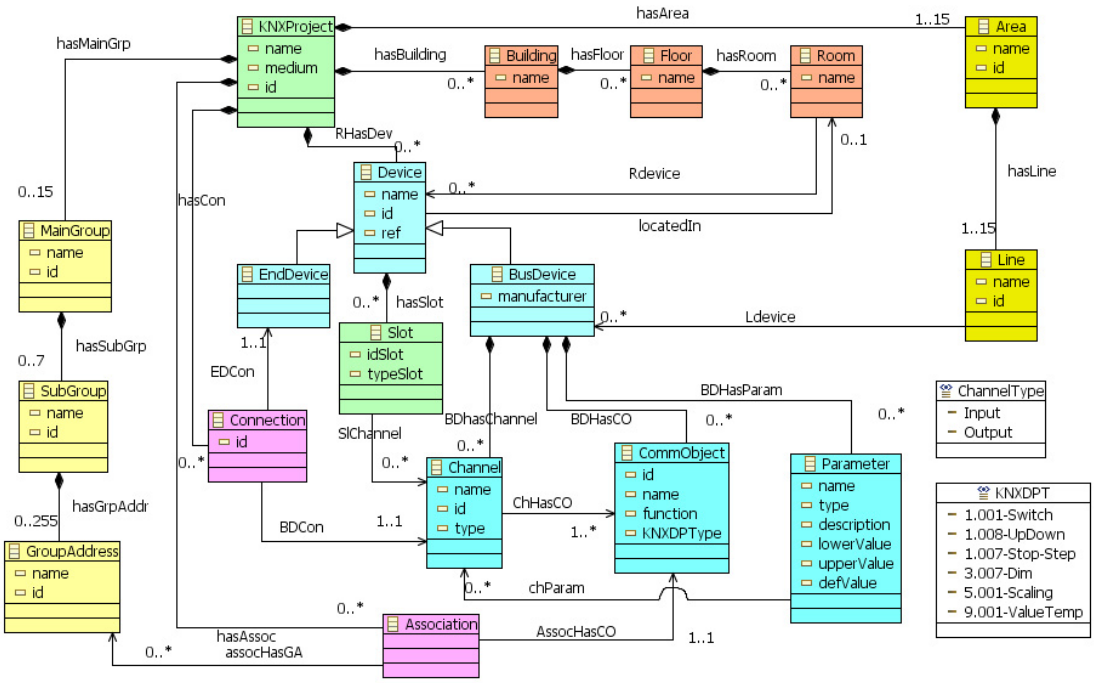


Figura 5-34. Metamodelo de nivel PSM para KNX/EIB modificado con soporte para *Slots*.

Siguiendo con la estrategia planteada en el apartado 5.4.1 para los niveles CIM – PIM, se expondrán las transformaciones PIM – PSM más representativas en lenguaje natural para facilitar su comprensión. Estas transformaciones tendrán como modelos de partida el modelo de nivel PIM obtenido a partir de las transformaciones desde el modelo CIM y el modelo de configuración de *slots* de los dispositivos.

En la Tabla 5-2 se resumen, en lenguaje natural, las principales reglas de transformación entre los metamodelos de nivel PIM y PSM.

ID	Descripción de la regla de transformación
1.	En el modelo del nivel PSM se crearán los objetos área con identificador *id=1* y línea con *id=1* para para establecer una topología inicial de dispositivos.
2.	Para cada componente *SimpleComponent* del modelo PIM se creará un dispositivo *Device* en el modelo PSM dentro del área y la línea creadas en la regla 1. El atributo *id* tomará valores incrementales partiendo de 1, y el atributo *name* tomará el valor del atributo *tag2* seguido del literal ". " y el atributo *tag1* del *SimpleComponent* del modelo PIM.
2a.	Si el *SimpleComponent* del modelo PIM tiene el atributo *tag3=true* se creará un *EndDevice*.
2b.	Si el *SimpleComponent* del modelo PIM tiene el atributo *tag3=false* se creará un *BusDevice*.
3.	Las asociaciones entre puertos del modelo PIM mediante *PortLink* se transformarán en asociaciones entre canales u objetos de comunicación de la siguiente forma:
3a.	Para cada una asociación mediante un objeto *PortLink* con entre puertos de componentes teniendo uno de los *SimpleComp* el atributo *tag3=true* (es de tipo final), se creará un objeto *Connection* asociando los objetos *Channel* asociados a los *Slots* de los dispositivos implicados.
3b.	Para cada una asociación mediante un objeto *PortLink* que parta del puerto de uno o varios componentes al puerto o puertos de otro(s) componente(s) teniendo ambos *SimpleComp* el atributo *tag3=false* en el modelo PIM, se creará en el modelo PSM una nueva dirección de grupo *GroupAddress* (sin repetir identificadores) y un objeto *Association* entre el objeto *GroupAddress* y cada objeto de comunicación *CommObject* asociado al *Channel – Slot - SimpleComp* del otro extremo de la asociación *PortLink* en el modelo PIM.
3c.	A partir del tipo de servicio del nivel PIM (almacenado en el atributo *name* del objeto *Port*), el objeto *ServiceParam* asociado y del tipo *KNXDPT* en el modelo PSM se establecerá el tipo del objetos de comunicación *CommObject* que se enlazan para los servicios (o Puerto – Interfaz – Servicio) del modelo PIM. El tipo de asociación *IPort – Interface* en el modelo PIM determinará el atributo *type* del objeto *Channel* en el *Slot* del dispositivo del PSM: las asociaciones *Required* en el nivel PIM crearán un objeto *Channel* de tipo *Output* en el PSM, y las de tipo *Provided* en el PIM un objeto *Channel* de tipo *Input*. Los servicios se transforman a objetos de comunicación con una función determinada y un tipo de datos determinado (por ejemplo un servicio de tipo *switch* se transforma a un *CommObject* con tipo de dato *KNXDPT* 1.001)
4.	Los objetos *CompParam* del nivel PIM se transformarán en objetos de tipo *Parameter* en el nivel PSM. Los atributos *name, type y value* del *CompParam* en el modelo PIM servirán para establecer los valores de los atributos *name, type y value* de los objetos *Parameter* del modelo PSM.

Tabla 5-2. Reglas de transformación PIM - PSM más representativas expresadas en lenguaje natural.

Como se puede observar, se han realizado algunas simplificaciones con el fin de obtener un primer conjunto de reglas de transformación de fácil aplicabilidad que permitan verificar su correcto funcionamiento sobre modelos reales. Algunas de estas simplificaciones son:

- Se ha limitado el número máximo de dispositivos a 256, de manera que sólo sean necesarias una línea y un área en la topología de la instalación para albergarlos a todos. La ampliación del número de dispositivos pasaría por crear nuevas líneas (hasta 15) dentro de esta área o nuevas áreas (hasta 15), que a su vez podrían albergar hasta 15 líneas cada una.

- No se crea la estructura de edificio – planta - habitación (*Building – Floor – Room*). Esta estructura se creará en un futuro a partir de la clase autocontenida estancia (*Room*) del nivel CIM, para lo que será necesario definir las transformaciones CIM – PIM – PSM adecuadas. En este contexto se prevé la creación de una nueva vista del DSL de nivel CIM que permita ubicar los dispositivos en estancias reflejando la estructura física del entorno de la instalación (planos de planta de la vivienda o edificio).

Como trabajo inmediato en esta línea de investigación se propone la definición de las reglas mediante gramática de grafos siguiendo el enfoque planteado para las transformaciones CIM – PIM. Las reglas de transformación, una vez verificadas, se podrían integrar en el entorno de desarrollo Eclipse mediante EMT [Biermann 06], o bien expresarlas en un lenguaje formal de transformación como ATL [ATL 08].

Finalmente, para la generación de código para los dispositivos de esta tecnología se propone la utilización del enfoque y herramientas propuestas en el apartado 5.3.2 y de *Java Emitter Template* (JET) [JET 07]. JET es un subproyecto de EMF centrado en simplificar el proceso de generación automática de código a partir de plantillas. Las características del código hacen particularmente interesante el uso de un lenguaje de plantillas como JET, ya que se necesitarán patrones fijos parametrizables para crear líneas y áreas, insertar nuevos componentes al sistema, asignarles la dirección física, crear objetos de comunicación para asociaciones, establecer valores de parámetros, etc.

5.5 Conclusiones y Aportaciones a la Tesis

En este capítulo se ha presentado una metodología basada en el enfoque dirigido por modelos como alternativa para resolver los problemas encontrados en el desarrollo tradicional de software para sistemas domóticos.

Esta propuesta utiliza el enfoque MDA junto a un lenguaje visual y específico del dominio domótico como metodología de modelado, partiendo de un alto nivel de abstracción para describir la lógica de negocio en el nivel CIM. En el nivel PIM se ha utilizado un modelo de componentes, lo que facilita, por una parte, la confluencia con otros dominios de sistemas reactivos como las redes de sensores, los sistemas robóticos o los sistemas de visión artificial y reduce, por otra, la complejidad del modelo del sistema.

Tal como propone MDA, todo el proceso de desarrollo está dirigido por modelos conforme a los correspondientes metamodelos de cada nivel, completado con la definición de las transformaciones necesarias para hacer evolucionar el modelo desde

los niveles más altos de abstracción hasta la generación de código. En esta línea se han definido reglas de transformación del nivel CIM al PIM, primero en lenguaje natural, y a continuación utilizando un lenguaje formal de gramática de grafos. Esto ha permitido verificar la validez de dichas reglas de transformación mediante su aplicación a modelos reales.

Finalmente se ha propuesto un enfoque para abordar la generación de código para la tecnología KNX/EIB, una de las tareas más complejas debido a las singularidades asociadas a la programación de dispositivos domóticos comerciales. Para ello, se ha diseñado un metamodelo del PSM KNX/EIB y se han definido las herramientas que soportan la interacción con la herramienta de programación de esta tecnología (ETS). Aunque el conjunto de transformaciones PIM-PSM es incompleto, con el aporte realizado se demuestra la viabilidad de la propuesta metodológica. Esta aportación permitirá completar el ciclo de desarrollo de sistemas domóticos como trabajo inmediato en el Grupo de Investigación DSIE.

Con esta metodología se contribuye a la evolución del desarrollo de sistemas domóticos simplificando notablemente el proceso de desarrollo, se proporcionan primitivas intuitivas, se aumenta el nivel de abstracción, se reduce la complejidad y se incrementa la calidad del software obtenido, aunque esto suele ser difícil de justificar. Al menos, hay que tener en mente que al sistematizar la generación de código se conseguirá el mismo nivel de exigencia que hace treinta años se consiguió con los compiladores de lenguajes de alto nivel.

CAPITULO 6

Conclusiones y Trabajos Futuros

En el presente apartado se hace una recapitulación de los resultados obtenidos en cada uno de los capítulos anteriores y de las aportaciones realizadas. A continuación se exponen los resultados concretos conseguidos durante su realización y, para finalizar, se sugieren trabajos futuros de investigación relacionados con la materia aquí tratada.

6.1 Conclusiones

La domótica es una disciplina reciente que integra numerosos servicios y sistemas relacionados con la gestión de viviendas y edificios. En el desarrollo de los sistemas domóticos se ven involucradas diversas materias como la automatización, tecnologías de la información, gestión de redes o la programación de microprocesadores.

En los últimos años han surgido numerosos sistemas y estándares domóticos, y aunque siguen evolucionando, los recientes procesos de convergencia de estándares demuestran que se está alcanzando cierta madurez, lo que permite dirigir un mayor esfuerzo a su mejora, más que a la búsqueda de nuevas soluciones tecnológicas o arquitectónicas. En este sentido, los sistemas mejor posicionados son Konnex, resultado del proceso de convergencia de los principales sistemas europeos, y Lonworks, una de las tecnologías con mayor implantación a nivel mundial. Este aspecto se ha tenido en cuenta a la hora de seleccionar la plataforma inicial sobre la que realizar las aplicaciones. La evolución en las tecnologías domóticas ha sido paralela a un crecimiento casi exponencial de la demanda de estos sistemas en el sector doméstico y terciario.

En la actualidad, el proceso de diseño en el campo de la domótica es similar al empleado en otros sistemas reactivos que interactúan con el entorno, como los de automatización y control industrial, la robótica o los sistemas de inspección automatizados. En todos ellos es necesaria la intervención de un especialista del dominio que tiene una amplia experiencia en la plataforma sobre la que se realizará la implementación. Además, en la mayoría de los casos se realiza el diseño prácticamente desde cero, y se requiere un gran esfuerzo para la generación del código en el lenguaje de programación que se vaya a utilizar, conduciendo a soluciones a medida que rara vez son reutilizadas. Estos y otros muchos problemas plantean la necesidad de una mejora en el enfoque utilizado en el proceso de desarrollo.

En el caso particular de los sistemas domóticos, el componente software es importante, puesto que han de programarse dispositivos o nodos (sensores, actuadores y controladores) en una arquitectura centralizada o distribuida. Por ello, la aplicación de técnicas de Ingeniería del *Software* contribuye a mejorar el proceso de desarrollo de estos sistemas.

En esta Tesis ha sido clave la utilización de un planteamiento que está revolucionando la Ingeniería del *Software* en los últimos años: el desarrollo dirigido por modelos (MDE). Esta tecnología es relativamente reciente (se empieza a plantear en el año 2000), y se ha visto impulsada por la aportación del OMG con su propuesta MDA y los numerosos estándares y herramientas que están haciendo posible su aplicación en la actualidad. Su aplicación ha permitido definir una metodología que abarque todo el ciclo de vida en el desarrollo de sistemas domóticos, mejorando el proceso en su conjunto.

Otro aspecto importante ha sido la utilización de un lenguaje específico de dominio, que incrementa el potencial del enfoque dirigido por modelos aportando una infraestructura para la captura de requisitos con un alto nivel de abstracción y de forma independiente de la plataforma. Este lenguaje se ha utilizado en el nivel independiente de la computación, el más alto en la propuesta MDA. A partir de él se definen las transformaciones de modelo a modelo necesarias para obtener un modelo de componentes intermedio que garantice la compatibilidad con otros sistemas reactivos como las redes de sensores, visión o la robótica. Finalmente se definen las transformaciones para obtener modelos de una plataforma específica, a partir de los cuales se pueden aplicar transformaciones de modelo a texto para completar el ciclo y obtener una serie de plantillas compatibles con las aplicaciones domóticas existentes.

La presente Tesis Doctoral complementa los trabajos realizados en el Grupo de Investigación DSIE en otros dominios de sistemas reactivos, a la vez que permite la confluencia con todos ellos gracias a la utilización de modelo común de componentes.

En lo referente a las herramientas utilizadas (concretamente Eclipse y los *plugins* disponibles), si bien es el único entorno que proporciona el soporte necesario para realizar un diseño dirigido por modelos, se encuentra todavía en fase de desarrollo. La gran cantidad de *plugins* existentes, añadido al hecho de que las diferentes versiones no son totalmente compatibles entre sí, complica en gran medida la tarea de obtener una versión "estable" para todas sus versiones. A esto hay que añadir la complejidad añadida por algunos de los *plugins*, como GMF, y la dificultad para conseguir determinadas características en los aspectos gráficos, que deben implementarse a través de la modificación del código Java generado. No obstante, gracias a su naturaleza abierta, Eclipse es una herramienta con un enorme potencial que se está convirtiendo en un estándar de uso entre los miembros de la comunidad software.

6.2 Aportaciones

A continuación se enumeran las principales aportaciones y características más sobresalientes de la presente Tesis Doctoral. En opinión del autor, este trabajo cumple con los objetivos presentados en el capítulo de introducción, aunque es inevitable que se produzcan algunas variaciones con la evolución del trabajo de investigación.

- Se ha realizado un estudio pormenorizado del dominio de aplicación de los sistemas domóticos, revisando las características de las principales tecnologías (la mayoría recogidas por estándares internacionales). La comprensión del dominio y de las tecnologías existentes es fundamental para entender el proceso de diseño que se sigue para desarrollar sistemas domóticos y los problemas que conlleva.
- Se ha justificado el interés de los sistemas domóticos en la sociedad actual, por un lado, por la demanda creciente de los servicios que ofrece (confort, seguridad, gestión energética y comunicaciones), y por otro, por el crecimiento que está teniendo la instalación de este tipo de sistemas en los últimos años.

- Se han planteado los principios básicos del paradigma de desarrollo dirigido por modelos y los beneficios que puede aportar para solucionar las carencias más importantes en el desarrollo tradicional de sistemas reactivos, como las mejoras en flexibilidad, reutilización y calidad de los diseños realizados. Asimismo, se han descrito las herramientas disponibles para dar soporte a la propuesta de diseño de sistemas domóticos dirigido por modelos.
- Se han revisado los trabajos existentes en el ámbito MDA – domótica, lo que ha permitido establecer algunos de los fundamentos empleados en este trabajo de Tesis.
- Se ha definido un lenguaje específico del domino domótico con una sintaxis concreta de tipo gráfico y el mayor nivel de abstracción posible. Asimismo se ha diseñado un metamodelo para dar soporte a su sintaxis abstracta, y las restricciones necesarias para asegurar la corrección de las expresiones definidas a partir del lenguaje. Este lenguaje proporciona al usuario un entorno visual donde definir los requisitos del sistema, quedando descrita toda la lógica de negocio desde una perspectiva independiente de computación e independiente de la plataforma. Para la definición del lenguaje se han presentado los enfoques y las características más importantes de los DSLs y se han revisado las propuestas existentes para el dominio de la domótica, analizando sus ventajas e inconvenientes.
- Se ha establecido una nueva metodología para resolver los problemas encontrados en el desarrollo tradicional de software para sistemas domóticos. Esta metodología, bautizada con el nombre de HAbitATION, utiliza de forma conjunta un lenguaje específico de dominio y el enfoque de desarrollo dirigido por modelos, en concreto la propuesta MDA del OMG. Para ello se han realizado las siguientes aportaciones:
    - Se ha utilizado el DSL para describir la lógica de negocio en el nivel CIM con un alto grado de abstracción.
    - En el nivel independiente de plataforma (PIM) se ha utilizado un modelo de componentes (V^3Studio) lo que facilita, por una parte, la confluencia con otros dominios de sistemas reactivos en el ámbito del Grupo de Investigación DSIE y reduce, por otra, la complejidad del modelo del sistema.
    - Se ha propuesto un enfoque para abordar la generación de código para la tecnología KNX/EIB, una de las tareas más complejas debido a las singularidades asociadas a la programación de dispositivos domóticos comerciales. Para ello se ha diseñado un metamodelo de nivel PSM para KNX/EIB y se han definido las herramientas que soportan la interacción con la herramienta de programación de esta tecnología (ETS).
    - Se han definido transformaciones necesarias para hacer evolucionar el modelo desde los niveles más altos de abstracción hasta los más bajos dependientes de la plataforma. En esta línea se han definido todas las reglas de transformación del nivel CIM al PIM primero en lenguaje natural y a continuación utilizando un lenguaje formal de gramática de grafos. Esto ha permitido verificar la validez de dichas reglas de transformación mediante su aplicación a modelos

reales. Además, se han propuesto las principales reglas de transformación, en lenguaje natural, para obtener modelos específicos de la plataforma KNX/EIB.

- Se han implementado las herramientas para el desarrollo de sistemas domóticos utilizando Eclipse junto con los diferentes *plugins* disponibles (EMF, GMF, etc.) para dar soporte al desarrollo dirigido por modelos.

- Finalmente, se ha desarrollado un caso de estudio utilizando el DSL para demostrar su utilidad en la definición de una aplicación domótica, en concreto la implementación de una serie de servicios en una sala de reuniones. Este caso de estudio se describe en el Anexo A, donde se detalla la funcionalidad y los dispositivos domóticos instalados en las dependencias de la Universidad.

Con esta metodología se contribuye a la evolución del desarrollo de sistemas domóticos simplificando notablemente el proceso de desarrollo, se proporcionan primitivas intuitivas, se aumenta el nivel de abstracción, se reduce la complejidad y se incrementa la calidad del software obtenido.

6.3 Divulgación de Resultados

Los resultados obtenidos como parte de la realización de este trabajo de Tesis, realizado en el seno del Grupo de Investigación DSIE de la Universidad Politécnica de Cartagena, han dado lugar a las publicaciones que se enumeran a continuación.

- **Revistas**

  - Jiménez, M.; Rosique, MF.; Sánchez, P.; Álvarez, B.; Iborra, A.: "HAbitATION: a Domain Specific Language for Home Automation". IEEE Software (en proceso de revisión).
  - Jiménez, M.; Rosique, MF.; Sánchez, P.; Álvarez, B.; Iborra, A.: "A Domain Specific Language for developing Home Automation Applications following a Model Oriented Approach.". Science of Computer Programming, Elsevier (en proceso de revisión).
  - Jiménez, M; Rosique, M.F.; Sánchez, P.; Álvarez, B.; Iborra, A.: "Software Orientado a Modelos para Aplicaciones Domóticas". Automática e Instrumentación, vol. 395, pp. 110-115. ISSN 0213-3113, 2008.
  - Rosique, M.F.; Jiménez, M.; Sánchez, P.; Álvarez, B.; Iborra, A.: "Desarrollo de Aplicaciones Domóticas Dirigido por Modelos". Revista de las I Jornadas de Introducción a la Investigación de la UPCT, pp. 6-8. ISSN 1888-8356, 2008.
  - Jiménez, M.; Vera, J.A.; Losilla, F.; Meseguer, P.J.: "Redes de Sensores y Actuadores (WSAN) en domótica", Revista Telecoforum, vol. 5, pp. 59-62. ISSN 1698-2924, 2007.

- **Congresos nacionales**

  - Jiménez, M.; Vera, J.A.; Losilla, F.; Sánchez, P.; Iborra, A.: "Experiencia en la integración de Redes de Sensores y Actuadores (WSAN) en domótica". En: Actas del XIV Seminario Anual de Automática, Electrónica Industrial e Instrumentación *(SAAEI'07).* ISBN 978-968-9182-52-8, 2007. Puebla, México, 2007.

- **Reuniones científicas**

  - I Jornadas de Trabajo META: Models, Environments, Transformations and Applications. Ciudad Real, 2007.
  - II Jornadas de Trabajo META: Models, Environments, Transformations and Applications. Cartagena, 2008.

- **Informes Técnicos**

  - Jiménez, M.; Álvarez, B.; Sánchez, P.; Iborra, A.: "Desarrollo de nuevos productos domóticos y domotización de una sala de reuniones". Informe técnico proyecto MEDWSA (DT-MEDWSA-03), 24 páginas, 2007.
  - Rosique, M.F.; Jiménez, M.; Sánchez, P.: "Modelado de Sistemas Domóticos". Informe técnico proyecto MEDWSA (DT-MEDWSA-09), 17 páginas, 2007.
  - Rosique, M.F.; Jiménez, M.; Sánchez, P.: "Sistema Domótico para una Sala de Juntas: Uso de un Lenguaje Específico de Dominio". Informe técnico proyecto MEDWSA (DT-MEDWSA-10), 14 páginas, 2008.
  - Roca, P.J.; Rosique, M.F.; Jiménez, M.; Sánchez, P.: "Instalaciones previas ITTools - WIBUKEY para generación de macros con ETS3". Informe técnico proyecto MEDWSA (DT-MEDWSA-11), 24 páginas, 2008.
  - Rosique, M.F.; Jiménez, M.; Sánchez, P.: "Definición de Metamodelos, Herramientas y Transformaciones para Aplicaciones Domóticas". Informe técnico proyecto MEDWSA (DT-MEDWSA-12), 117 páginas, 2008.
  - Rosique, M.F.; Jiménez, M.; Sánchez, P.: "Visual Languages for Home Automation: a Model Driven Approach". Informe técnico proyecto MEDWSA (DT-MEDWSA-13), 24 páginas, 2008.
  - Jiménez, M.; Rosique, M.F.; Sánchez, P.: "Definición de Transformaciones de Modelos CIM-PIM en el Dominio Domótico". Informe técnico proyecto MEDWSA (DT-MEDWSA-14), 30 páginas, 2008.

- **Marco de desarrollo**

Además de las publicaciones enumeradas, el trabajo realizado en esta Tesis Doctoral se ha desarrollado en el marco del proyecto META – subproyecto MEDWSA (*Marco conceptual y tecnológico para el desarrollo de software de sistemas reactivos*) de la Comisión Interministerial de Ciencia y Tecnología (CICYT TIN2006-15175-C05-02).

6.4 Trabajos Futuros

Cuando se abre una nueva línea de investigación resulta inevitable abrir un amplio abanico de nuevos frentes de trabajo, consecuencia del conocimiento adquirido durante su desarrollo. En este último apartado se recogen las principales líneas de investigación que se encuentran en marcha en el marco del proyecto MEDWSA, así como otras en las que resultaría de interés seguir profundizando en el futuro.

En lo referente a trabajos actualmente en marcha en el DSIE:

- Completar la integración de las reglas de transformación CIM-PIM en el entorno Eclipse mediante el *plugin* EMT. Dicha integración no se llevó a la práctica durante el desarrollo de este trabajo debido a su no disponibilidad temporal por modificaciones en su funcionamiento. Por ello se optó por definir y validar las reglas con la interfaz del motor de transformación de grafos AGG, que son exportables de manera inmediata al *plugin* EMT, que utiliza el mismo motor de transformaciones.
- Completar el conjunto de transformaciones PIM-PSM utilizando las herramientas propuestas en esta Tesis. Se está trabajando asimismo en la definición de las plantillas JET de generación de código para la plataforma KNX/EIB, que se validarán en el caso de estudio definido con el DSL.
- Desarrollar el DSL para la creación de nuevas unidades funcionales que no se encuentran definidas en el catálogo para dar un soporte más flexible a la variabilidad del catálogo.
- Crear una vista de plano para las aplicaciones. El metamodelo propuesto para el nivel CIM contiene las clases necesarias para soportar la ubicación de unidades funcionales en estancias y, en estos momentos, se está trabajando en su implantación dentro de la herramienta.
- Añadir catálogos de requisitos para establecer un repositorio de soluciones. En esta línea se está colaborando con los autores del entorno REMM-Studio (*Requirements Engineering Metamodel*) para el modelado de requisitos reutilizables. REMM ha sido desarrollado por Grupo de Ingeniería del *Software* del Departamento de Informática y Sistemas de la Universidad de Murcia dentro del proyecto META - subproyecto DEDALO (*Desarrollo de sistEmas de caliDad bAsado en modeLos y requisitOs*) de la Comisión Interministerial de Ciencia y Tecnología (TIC2006-15175-C05-03).

Entre las cuestiones que han quedado abiertas, y que resultará interesante abordar igualmente en el futuro, cabe mencionar:

- Desarrollar metamodelos y transformaciones para otras plataformas de nivel PSM. Entre las posibles tecnologías sería interesante realizar este trabajo para Lonworks.

- Realizar análisis de trazabilidad para comprobar el cumplimiento de determinados requisitos (por ejemplo "*se ahorra energía en iluminación", "se garantiza la seguridad de bienes y personas"*). Esto implicaría utilizar herramientas de transformación que permitan realizar este tipo de análisis.

- Validar la usabilidad y capacidad expresiva del DSL definiendo métricas, estrategias y grupos de muestreo (estudiantes con conocimientos de domótica, profesionales, etc.).

- Desarrollar entornos de animación generados a partir del DSL para facilitar la validación de las aplicaciones sin necesidad de programar los dispositivos reales.

ANEXO A

Caso de Estudio

En este anexo se describe un caso de estudio, en el ámbito de la domótica, consistente en la automatización de una sala de juntas o reuniones utilizando el DSL presentado en el capítulo 4. El caso que se describe ha permitido, por un lado, validar el funcionamiento del DSL en una aplicación real y, por otro, constituye un referente de partida en las líneas de investigación ya en marcha en el DSIE para aplicar la metodología propuesta en este trabajo de Tesis.

El anexo se divide en cinco secciones. En primer lugar se introduce el caso de estudio. A continuación se presentan los elementos de interacción con el sistema y los servicios que se ofrecerán. El siguiente apartado detalla los casos de uso para los actores del sistema. Finalmente, la última sección describe los modelos del caso de estudio representados con el DSL.

A.1 Introducción

El ámbito de aplicación de los sistemas domóticos es muy amplio, abarcando desde pequeñas instalaciones en viviendas hasta grandes sistemas como los instalados en hoteles, hospitales o edificios de oficinas. En este anexo se presenta un caso de estudio de un sistema con la complejidad y tamaño adecuados para validar el DSL, siendo a su vez viable en el seno del Grupo de Investigación DSIE. Las características del DSL (basado en un catálogo de elementos reutilizables) y de la metodología (dirigida por modelos con definición de requisitos de forma independiente de la plataforma) facilitan su extensión a sistemas mayores.

En la Figura A-1 se muestra el plano de la sala en la que se van a implementar los servicios de automatización. Se trata de la sala de juntas del DSIE, utilizada para diferentes actividades como reuniones del personal, seminarios y diversos tipos de presentaciones. La finalidad del sistema domótico es lograr ofrecer una serie de servicios integrados para lograr los cuatro objetivos básicos para cualquier instalación de este tipo: confort, seguridad, ahorro energético y comunicaciones. Con este fin se describen, en primer lugar, los elementos a controlar para lograr la interacción con el usuario y, a continuación, los servicios ofrecidos, que se encuentran descritos con detalle en forma de casos de uso en el apartado A.4. En la Figura A-2 ilustra algunos de los dispositivos comerciales instalados.

A.2 Elementos de Interacción con el Sistema

Los elementos con los que va a interactuar el sistema son de dos tipos: sensores o entradas, que recogen eventos del entorno (interacción del usuario, parámetros ambientales, etc.), y actuaciones o salidas, que cambian de estado como consecuencia de la implementación de los servicios. La Figura A-1 detalla la ubicación de los elementos utilizando la representación gráfica de las unidades funcionales del DSL para aplicaciones domóticas. A continuación se detallan dichos elementos:

- Iluminación. Se regulan los seis puntos de luz de la sala (LDM-1 a LDM-6).
- Motorizaciones. La pantalla de proyección y la persiana veneciana disponen de motorizaciones (SHM1 y SC-1) para el control de su posición.
- Seguridad. Para la gestión de la seguridad se instalarán los siguientes dispositivos:
  - Detectores de movimiento en techo para monitorizar la presencia de personas en la sala (PIR3-1 y PIR3-2).
  - Detectores de humo e inundación (FD-1, WD-1 y WD-2).
- Climatización. Se controlarán los dos climatizadores de tipo *fancoil*, que disponen de tres velocidades de ventilación (Fan3-1) y una electroválvula (VO-1) para el paso de fluido de climatización.

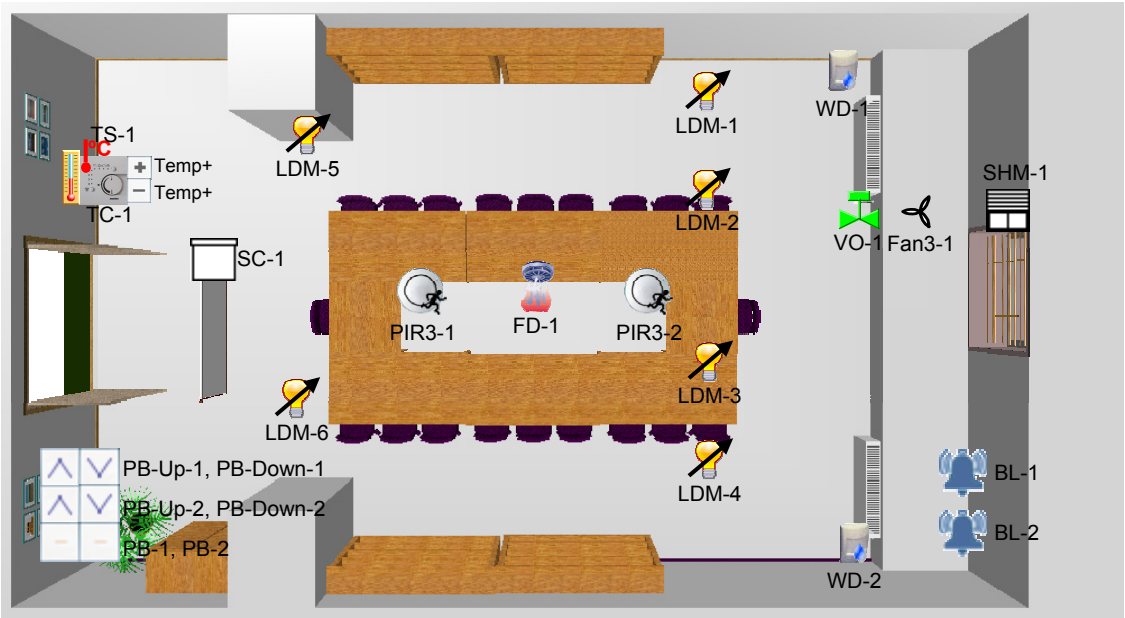


Figura A-1. Plano de la sala de juntas con elementos de interacción con el sistema domótico.

Figura A-2. Algunos dispositivos comerciales instalados en la sala de juntas. (a) y (b) Pantalla y persiana veneciana motorizadas. (c) Termostato. (d) Pantalla táctil. (e) Botonera. (f) Detectores de movimiento y fuego.

A.3 Resumen de Servicios

A continuación se resumen los servicios a prestar categorizados según las funciones típicas integradas en los sistemas domóticos:

- **Gestión energética:**
  - Iluminación. La iluminación se desactiva automáticamente una vez transcurrido un tiempo programado sin presencia en la sala.
  - Climatización. Se definen tres modos de climatización:
    - Confort (o presencia): la temperatura de consigna para la climatización es la deseada por el usuario.
    - Ausencia: en invierno se reduce la temperatura de consigna y en verano se incrementa para conseguir ahorro energético, pero sin apagar la climatización.
    - Noche: en el horario nocturno se realizan los mismos cambios que en el modo ausencia, pero con una diferencia térmica menor respecto de la temperatura de confort.

    Los cambios de modo serán automáticos por detección de presencia (entre modos ausencia y confort) y por programación horaria (modo noche).

- **Confort**:
  - Iluminación:
    - Será controlable desde pulsadores de encendido/regulación.
    - El apagado será automático por ausencia prolongada.
  - Climatización:
    - Cambio automático de modos por detección de presencia y tramos horarios.
  - Escenas / funciones centrales. El administrador de la instalación podrá definir escenas o ambientes que impliquen una secuencia de actuaciones. Se establecerán las siguientes:
    - Presentación: se baja la pantalla de proyección, se regula la iluminación a un 20%.
    - Reunión: se regula toda la iluminación a un 100% y se sube la persiana veneciana.
- **Seguridad**. Implica dos tipos de servicios: seguridad ante intrusión y de bienes y personas (alarmas técnicas):
  - Seguridad de intrusión: en caso de detección de presencia en zona de acceso cuando la alarma está activada, se generará un aviso acústico y se notificará al usuario.

- Seguridad de bienes y personas: en caso de detección de inundación se apaga la calefacción (única fuente de fugas de agua en la sala), se generará un aviso acústico y se notifica al usuario. En el caso de detección de humo, también se le notificará.

- **Comunicación**:

  Con el fin de controlar el estado de la instalación y enviar las incidencias que en ésta se produzcan, se dispondrá de una pasarela con capacidad de envío de mensajes a teléfonos móviles (SMS).

A.4 Diagrama de Casos de Uso

Para recoger de modo formal los requisitos de la instalación, en este apartado se representan los diagramas de casos de uso detallando todos los servicios que se ofrecen en la instalación (véase la Figura A-3 y la Figura A-4). Los actores pueden ser de dos tipos: usuario y administrador del sistema. El actor habitual será el usuario, que representa a las personas que hacen uso de la sala y sus servicios (es decir, personal del Grupo de Investigación). El administrador es el experto del dominio encargado de programar la instalación y realizar modificaciones en su dinámica de funcionamiento, así como en los elementos implicados en la dinámica de cada uno de los servicios. El administrador realizará estas modificaciones utilizando las herramientas de programación adecuadas ejecutadas en un ordenador personal con conexión local o remota al sistema.

Figura A-3. Diagrama de casos de uso para el actor usuario.

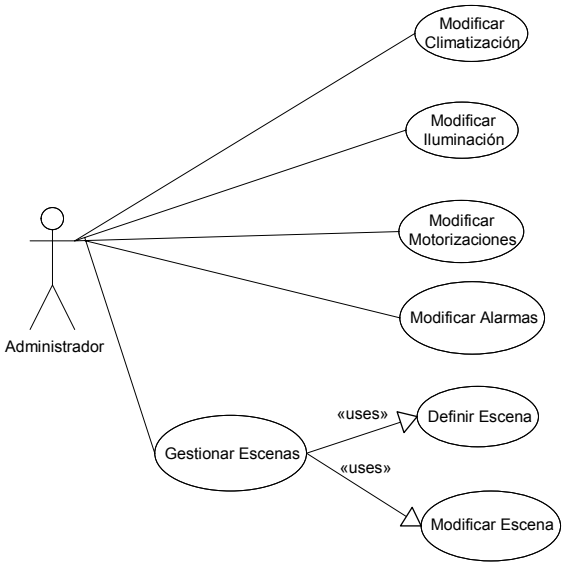


Figura A-4. Diagrama de casos de uso para el actor administrador.

A continuación se describen con detalle cada uno de los casos de uso para el actor usuario y el actor administrador.

A.4.1 Casos de Uso para el Usuario

Nombre del caso de uso: Modificar Tª confort
Paquete donde se ubica: GestionClima.
Resumen: el usuario modifica la temperatura del modo confort.
Actores: usuario (primario).
Precondiciones: el usuario tiene acceso a un termostato, pantalla táctil o visualización *web* con opción de modificación de temperatura de confort.
Descripción:

1. El usuario desea modificar la temperatura de confort (tiene frío o calor).
2. El usuario se aproxima a un termostato.
3. El usuario realiza una pulsación sobre uno de los botones para aumentar o disminuir la temperatura de consigna en 1˚C.
4. Se repite el paso 3 hasta conseguir la temperatura deseada.

Alternativas:

2a. Termostato analógico, accionamiento sobre mando giratorio.
2b. Pantalla táctil: entrar en modo temperatura y realizar las mismas acciones.
2c. Visualización web: entrar en modo temperatura y realizar las mismas acciones.

Postcondición: la nueva temperatura de confort se establece como punto de consigna en el termostato para la regulación de aparatos de climatización.
Requisitos especiales: no necesarios.

Nombre del caso de uso: Modificar Modo Clima.
Paquete donde se ubica: Climatización.
Resumen: se modifica el modo de funcionamiento de la climatización.
Actores: usuario (primario).

Precondiciones: la instalación dispone de termostato o controlador de temperatura y de detectores de presencia.
Descripción:

1. El usuario entra en la sala: el termostato cambia a modo confort.
3. No hay usuarios en la sala: transcurridos cinco minutos sin presencia el termostato cambia a modo ausencia.
4. En horario nocturno (00:00 a 8:00) el termostato pasa a modo noche.

Alternativas:
Postcondición: el nuevo modo de funcionamiento queda establecido y se utiliza la temperatura asociada a dicho modo para la regulación de aparatos de climatización.
Requisitos especiales:
Preferible visualización de temperatura en display digital.
Recuperación de valor establecido en caso de fallo de tensión.
El administrador fija la variación de la temperatura para los modos ausencia y noche.

Nombre del caso de uso: Modificar Iluminación.
Paquete donde se ubica: GestionIlluminacion.
Resumen: el usuario enciende, apaga o regula los puntos de iluminación de la sala.
Actores: usuario (primario).
Precondiciones: el usuario tiene acceso a un pulsador, pantalla táctil o visualización web con opción de modificación de T confort.
Descripción:

1. El usuario desea encender, apagar o regular un punto de luz.
2. El usuario se aproxima a un pulsador de iluminación.
3. El usuario realiza una pulsación corta sobre el botón on/incrementar para encender la luz o sobre el botón off/decrementar para apagarla.
4. El usuario realiza una pulsación larga sobre el botón on/incrementar para aumentar el nivel luminoso o sobre el botón off/decrementar para decrementarlo.
5. Se repiten los pasos 3 y 4 hasta llegar al estado deseado.

Alternativas:

2a. Pantalla táctil: entrar en menú de iluminación de la estancias y realizar las mismas acciones. La pantalla debe indicar el estado de la luz.
2b. Visualización web: entrar en menú de iluminación de la estancias y realizar las mismas acciones. La visualización debe indicar el estado de la luz.
2d. La iluminación se enciende automáticamente por detección de presencia en la sala y se apaga tras cinco minutos de ausencia.

Requisitos especiales:
El administrador establece velocidad de regulación y valor de inicialización en caso de fallo de la tensión.
Para el encendido/apagado automático son necesarios detectores de presencia.
El administrador puede asignar las acciones a una sola tecla en modo alternativo.

Nombre del caso de uso: Regular Motorización.
Paquete donde se ubica: GestionMotorización.
Resumen: el usuario sube, baja o regula la posición de un dispositivo motorizado.
Actores: usuario (primario).

Precondiciones: el usuario tiene acceso a un pulsador, pantalla táctil o visualización web con opción de actuación sobre motorizaciones.
Descripción:

1. El usuario desea subir, bajar o regular el elemento motorizado.
2. El usuario se aproxima a un pulsador de control de motorización.
3. El usuario realiza una pulsación larga sobre el botón arriba para iniciar el movimiento ascendente del elemento motorizado o bien sobre el botón abajo para iniciar el movimiento descendente.

3.1. El usuario realiza una pulsación corta sobre cualquier tecla para detener el movimiento iniciado.

4. El usuario realiza una pulsación corta sobre el botón arriba para indicar orden de subir un tramo o sobre el botón abajo para indicar orden de bajar un tramo.
5. Se repiten los pasos 3 y 4 hasta llegar a la posición deseada.

Alternativas:

2a Pantalla táctil: entrar en menú de motorizaciones y realizar las mismas acciones. La pantalla puede indicar la posición del dispositivo motorizado.

2b. Visualización web: entrar en menú de motorizaciones y realizar las mismas acciones. La visualización puede indicar la posición del dispositivo motorizado.

Requisitos especiales:
El administrador establece:

- Tiempos para distinguir entre pulsación corta y larga.
- Tiempo para paso de subida o bajada.
- Tiempo de actuación sobre el motor para subida o bajada total.
- Valor de inicialización en caso de fallo de la tensión.

El administrador puede asignar las acciones a una sola tecla en modo alternativo.

Nombre del caso de uso: Regular Veneciana.
Paquete donde se ubica: GestionMotorización.
Resumen: el usuario sube, baja o regula la posición de una veneciana motorizada.
Actores: usuario (primario).
Precondiciones: el usuario tiene acceso a un pulsador, pantalla táctil o visualización web con opción de actuación sobre motorizaciones.
Descripción:

1. El usuario desea subir, bajar o regular la veneciana.
2. El usuario se aproxima a un pulsador de control de veneciana.
3. El usuario realiza una pulsación larga sobre el botón arriba para iniciar el movimiento ascendente de la veneciana o bien sobre el botón abajo para iniciar el movimiento descendente.

3.1. El usuario realiza una pulsación corta sobre cualquier tecla para detener el movimiento iniciado.

4. El usuario realiza una pulsación corta sobre el botón arriba para indicar orden de ajustar la lama o sobre el botón abajo para indicar orden de ajuste en la dirección contraria.
5. Se repiten los pasos 3 y 4 hasta llegar a la posición deseada.
6. El usuario se va.

Alternativas:

2a. Pantalla táctil: entrar en menú de motorizaciones y realizar las mismas acciones. La pantalla puede indicar la posición del dispositivo motorizado.

2b. Visualización web: entrar en menú de motorizaciones y realizar las mismas acciones. La visualización puede indicar la posición del dispositivo motorizado.

Requisitos especiales:

El administrador establece:

- Tiempos para distinguir entre pulsación corta y larga.
- Tiempo para paso de subida o bajada.
- Tiempo de actuación sobre el motor para subida o bajada total.
- Valor de inicialización en caso de fallo de la tensión.

El administrador puede asignar las acciones a una sola tecla en modo alternativo.

Nombre del caso de uso: Regular Pantalla de Proyección.

Paquete donde se ubica: GestionMotorización.

Resumen: El usuario sube, baja o regula la posición de la pantalla de proyección motorizada

Actores: usuario (primario).

Precondiciones: el usuario tiene acceso a un pulsador, pantalla táctil o visualización web con opción de actuación sobre la pantalla.

Descripción:

1. El usuario desea subir, bajar o regular la posición de la pantalla.
2. El usuario se aproxima a un pulsador de control de la pantalla.
3. El usuario realiza una pulsación larga sobre el botón arriba para iniciar el movimiento ascendente o bien sobre el botón abajo para iniciar el movimiento descendente.

3.1. El usuario realiza una pulsación corta sobre cualquier tecla para detener el movimiento iniciado.

4. El usuario realiza una pulsación corta sobre el botón arriba para indicar orden de subir un tramo o sobre el botón abajo para indicar orden de bajar un tramo.
5. Se repiten los pasos 3 y 4 hasta llegar a la posición deseada.
6. El usuario se va.

Alternativas:

2a. Pantalla táctil: entrar en menú de motorizaciones y realizar las mismas acciones. La pantalla puede indicar la posición del dispositivo motorizado.

2b. Visualización web: entrar en menú de motorizaciones y realizar las mismas acciones. La visualización puede indicar la posición del dispositivo motorizado.

Requisitos especiales:

El administrador establece:

- Tiempos para distinguir entre pulsación corta y larga.
- Tiempo para paso de subida o bajada.
- Tiempo de actuación sobre el motor para subida o bajada total.
- Valor de inicialización en caso de fallo de la tensión.

El administrador puede asignar las acciones a una sola tecla en modo alternativo.

Nombre del caso de uso: Iniciar Escena.

Paquete donde se ubica: Escenas.

Resumen: El usuario inicia la ejecución de una escena o función central.

Actores: Usuario (primario).

Precondiciones: El usuario tiene acceso a un pulsador, pantalla táctil o visualización web para iniciar la escena.

Descripción:

1. El usuario desea iniciar una escena preestablecida.
2. El usuario se aproxima a un pulsador de escenas.
3. El usuario realiza una pulsación corta sobre el pulsador.
4. Se reproduce la escena.
6. El usuario disfruta de la escena.

Alternativas:

2a. Pantalla táctil: entrar en menú escenas/funciones centrales y realizar las mismas acciones.

2b. Visualización web: entrar en menú de escenas/funciones centrales y realizar las mismas acciones.

Requisitos especiales:

El administrador establece los elementos y valores implicados en la escena.

Nombre del caso de uso: Notificar Alarma.
Paquete donde se ubica: Seguridad.
Resumen: Se envía una notificación de alarma al usuario.
Actores: Usuario (secundario).
Precondiciones: La seguridad está activada (sólo para el caso de alarma de intrusión).
Descripción:

1. Se produce la detección de un evento de seguridad (inundación, humo o intrusión).
2. El sistema genera un aviso acústico mediante una sirena.
3. Se notifica la incidencia al usuario mediante SMS.

Postcondición: el sistema está listo para procesar una nueva incidencia.
Alternativas:

3a. Se notifica la incidencia mediante correo electrónico.

3b. Se muestra la incidencia en una pantalla táctil y en la visualización web.

Requisitos especiales:

Si se detecta movimiento en la sala, esperar tiempo de entrada para notificar alarma de intrusión.

El administrador establece números de teléfono y direcciones de e-mail a los que enviar las incidencias.

Nombre del caso de uso: Desactivar Alarma.
Paquete donde se ubica: Seguridad.
Resumen: El usuario desactiva el sistema de alarma de intrusión.
Actores: Usuario (primario).
Precondiciones: La seguridad de intrusión está activada.
Descripción:

1. El usuario desea desactivar el sistema de seguridad de intrusión.
2. El usuario entra en la estancia y se aproxima al pulsador de activación/desactivación (con llave de acceso), introduce la llave y desactiva el pulsador.
3. Se desactiva la alarma.
4. Si se sobrepasa el tiempo de entrada o tres intentos fallidos se notifica la alarma.

Postcondición: el sistema de seguridad queda desactivado o se notifica una alarma de intrusión.
Alternativas: No hay alternativas.
Requisitos especiales:

La alarma de intrusión no se puede desactivar desde la visualización web (si la hay).

Nombre del caso de uso: Activar Alarma.
Paquete donde se ubica: Seguridad.
Resumen: El usuario activa el sistema de alarma de intrusión.
Actores: Usuario (primario).
Precondiciones: La seguridad de intrusión está desactivada.
Descripción:

1. El usuario desea activar el sistema de seguridad de intrusión.
2. El usuario se aproxima al pulsador de activación/desactivación (con llave de acceso), introduce la llave y activa el pulsador.
3. Se desactiva la alarma.

Postcondición: el sistema de seguridad queda activado.
Alternativas: no hay alternativas.
Requisitos especiales:
El administrador establece el tiempo de salida.

A.4.2 Casos de Uso para el Administrador

Nombre del caso de uso: Modificar Climatización.
Paquete donde se ubica: Gestión.
Resumen: el administrador modifica parámetros de funcionamiento de la climatización.
Actores: administrador (primario).
Precondiciones: el administrador dispone de las herramientas y el proyecto de la instalación (software residente en los dispositivos), y tiene permisos de modificación del software de los dispositivos.
Descripción:

1. El administrador conecta su ordenador a la instalación.
2. El administrador abre el proyecto de la instalación.
3. Realiza las modificaciones deseadas (eliminación de aparatos, inserción de nuevos, modificación de parámetros, modificación de asociaciones de dispositivos, etc.).

   Parámetros modificables: rango de Tª de confort que podrá establecer el usuario, tipo de control, actuaciones en caso de apertura de ventana, cambios automáticos de modo (ausencia/confort) por detección de movimiento, existencia del modo noche. Programaciones horarias para el modo noche.

Alternativas:

1a: la conexión se puede realizar en modo remoto a través de pasarela TCP/IP.

Postcondición: las modificaciones quedan grabadas en los dispositivos de climatización.
Requisitos especiales:

Nombre del caso de uso: Modificar Iluminación
Paquete donde se ubica: Gestión.
Resumen: el administrador modifica parámetros de funcionamiento de la iluminación.
Actores: administrador (primario).
Precondiciones: el administrador dispone de las herramientas y el proyecto de la instalación (software residente en los dispositivos), y tiene permisos de modificación del software de los dispositivos.

Descripción:

1. El administrador conecta su ordenador a la instalación.
2. El administrador abre el proyecto de la instalación.
3. Realiza las modificaciones deseadas (eliminación de aparatos, inserción de nuevos, modificación de parámetros, modificación de asociaciones de dispositivos, etc.).

Parámetros modificables: valor de luminosidad al encendido, tiempo de pulsación corta y larga, velocidad de regulación, apagados por ausencia.

Alternativas:

1a. La conexión se puede realizar en modo remoto a través de pasarela TCP/IP.

Postcondición: las modificaciones quedan grabadas en los dispositivos de iluminación.

Requisitos especiales:

Nombre del caso de uso: Modificar Motorizaciones

Paquete donde se ubica: Gestión.

Resumen: el administrador modifica parámetros de funcionamiento de las motorizaciones.

Actores: administrador (primario).

Precondiciones: el administrador dispone de las herramientas y el proyecto de la instalación (software residente en los dispositivos), y tiene permisos de modificación del software de los dispositivos.

Descripción:

1. El administrador conecta su ordenador a la instalación.
2. El administrador abre el proyecto de la instalación.
3. Realiza las modificaciones deseadas (eliminación de aparatos, inserción de nuevos, modificación de parámetros, modificación de asociaciones de dispositivos, etc.).

Parámetros modificables: tiempo de pulsación corta y larga, tiempos de recorrido de la motorización para movimiento total o ajuste de posición, posición en caso de fallo de suministro eléctrico. Programaciones horarias para automatismos.

Alternativas:

1a. La conexión se puede realizar en modo remoto a través de pasarela TCP/IP.

Postcondición: las modificaciones quedan grabadas en los dispositivos de motorización.

Requisitos especiales:

Nombre del caso de uso: Modificar Alarmas

Paquete donde se ubica: Gestión.

Resumen: el administrador modifica parámetros de funcionamiento de la alarma.

Actores: administrador (primario).

Precondiciones: el administrador dispone de las herramientas y el proyecto de la instalación (software residente en los dispositivos), y tiene permisos de modificación del software de los dispositivos.

Descripción:

1. El administrador conecta su ordenador a la instalación.
2. El administrador abre el proyecto de la instalación.
3. Realiza las modificaciones deseadas (eliminación de aparatos, inserción de nuevos, modificación de parámetros, modificación de asociaciones de dispositivos, etc.).

Parámetros modificables: números de teléfono para envío de incidencias, tiempos de entrada y de salida para alarma de intrusión. Acciones auxiliares en caso de disparo de alarma de intrusión (apagar luces, bajar persiana, etc.).

Alternativas:

1a. La conexión se puede realizar en modo remoto a través de pasarela TCP/IP.

3b. Si se dispone de servidor web se pueden modificar las direcciones de envío de correo electrónico.

Postcondición: las modificaciones quedan grabadas en los dispositivos de seguridad.

Requisitos especiales:

Nombre del caso de uso: Definir Escena.

Paquete donde se ubica: Gestión.

Resumen: el administrador define una nueva escena o función central.

Actores: administrador (primario).

Precondiciones: al administrador dispone de las herramientas y el proyecto de la instalación (software residente en los dispositivos), y tiene permisos de modificación del software de los dispositivos.

Descripción:

1. El administrador conecta su ordenador a la instalación.
2. El administrador abre el proyecto de la instalación.
3. Realiza las modificaciones deseadas.

Parámetros a definir: nombre de la escena, dispositivos involucrados, estado deseado para los dispositivos.

Alternativas:

1a. La conexión se puede realizar en modo remoto a través de pasarela TCP/IP.

Postcondición: las modificaciones quedan grabadas en los dispositivos de escenas.

Requisitos especiales:

Nombre del caso de uso: Modificar Escena.

Paquete donde se ubica: Gestión.

Resumen: El administrador modifica parámetros de funcionamiento de las escenas y funciones centrales.

Actores: Administrador (primario).

Precondiciones: El administrador dispone de las herramientas y el proyecto de la instalación (software residente en los dispositivos), y tiene permisos de modificación del software de los dispositivos.

Descripción:

1. El administrador conecta su ordenador a la instalación.
2. El administrador abre el proyecto de la instalación.
3. Realiza las modificaciones deseadas.

Parámetros modificables: nombre de la escena, dispositivos involucrados, estado deseado para los dispositivos.

Alternativas:

1a. La conexión se puede realizar en modo remoto a través de pasarela TCP/IP.

Postcondición: las modificaciones quedan grabadas en los dispositivos de escenas.

Requisitos especiales:

A.5 Desarrollo del Caso de Estudio con el DSL

En este apartado se describe el desarrollo del caso de estudio utilizando el DSL diseñado, dentro de la metodología propuesta, para la especificación de aplicaciones domóticas. Para ello, se ha partido del catálogo de unidades funcionales presentado en el capítulo 4.

A.5.1 Gestión de la Iluminación

En la Figura A-5 puede observarse el modelo del DSL para la iluminación de la sala. Se han empleado dos unidades funcionales pasivas del tipo *pulsador* (PB-1, PB-2) interconectadas cada una de ellas a través de un enlace de tipo *canal* (en color rojo) a una unidad funcional que actúa de controlador del tipo *entrada para regulación* (DMI-1, DMI-2) que requerirán el servicio de regulación lumínica necesario. En el extremo derecho de la Figura A-5 se representan las seis unidades pasivas de tipo *luz regulable* (LDM-1 a LDM-6) conectadas cada una de ellas a un controlador de tipo *salida para regulación*" (DMO-1 a DMO-6) que implementará el servicio de regulación lumínica. Finalmente se establecen una serie de enlaces entre las *entradas para regulación* y las *salidas para regulación* de manera que el pulsador PB-1 pueda regular las luces LDM-1,…, LDM-4 y el pulsador PB-2 las luces LDM-5, LDM-6.

Para el apagado automático, necesario para lograr el ahorro energético en iluminación, se han utilizado dos unidades pasivas del tipo *detector de presencia* (PIR3-1, PIR3-2) conectadas a un controlador del tipo *entrada de conmutación* que apagará las luces transcurridos cinco minutos (300 segundos) sin detectar presencia. El *temporizador* (TM-1) se encarga de medir dicho tiempo. Para ello, esta unidad funcional dispone del parámetro *TMItemp* para establecer el tiempo umbral de apagado y el parámetro *TMtype* para fijar el modo de comportamiento (*Off Delay*: retardo al apagado).

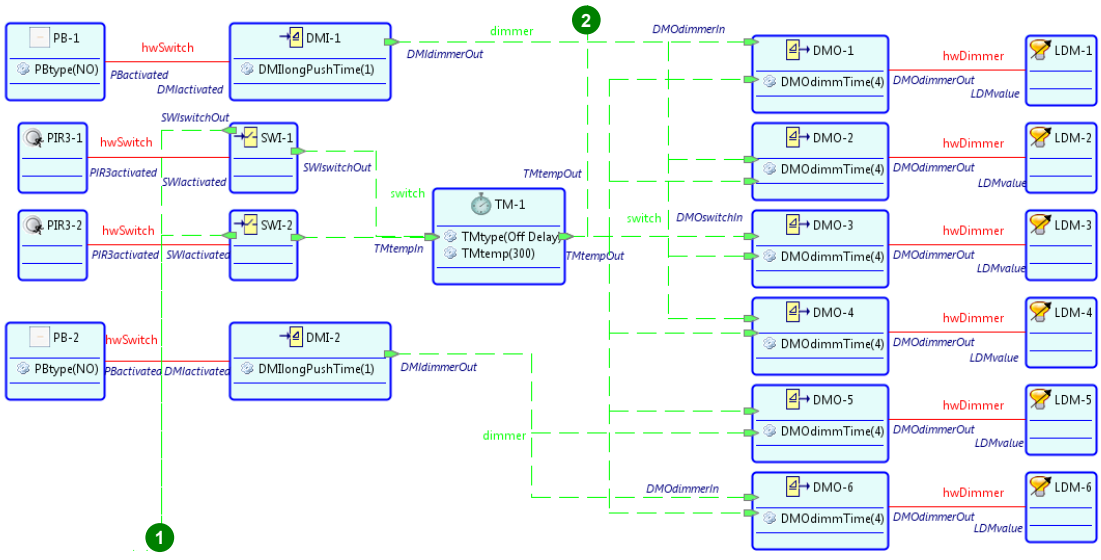


Figura A-5. Modelo de iluminación con el DSL.

Las etiquetas 1 y 2 indican enlaces de servicios de unidades funcionales que se encuentran en otras figuras del presente capítulo.

A.5.2 Control de Motorizaciones

Se controla el movimiento de subida y bajada de la veneciana y de la pantalla de proyección, así como la parada de movimiento de ambas. En la Figura A-6 se puede ver la parte del modelado realizado con el DSL para controlar estas motorizaciones, al igual que ocurría en la iluminación, los pulsadores (PB-Up-1, PB-Down-1, PB-Up-2, PB-Down-2) se conectarán por medio de canales de conexión a las unidades funcionales de entrada catalogadas como *Motorization In* (MTI-1 y MTI-2). Un pulsador será utilizado para la subida y otro para la bajada, en el otro extremo se tiene el controlador de salida (MTO-1) conectado a la persiana (SHM-1). La especificación para la pantalla de proyección (SC-1) es similar.

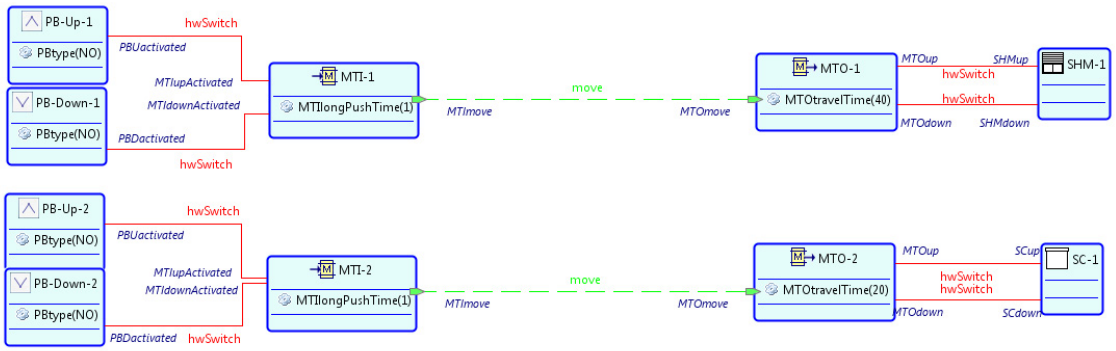


Figura A-6. Modelo de control de motorizaciones con el DSL.

A.5.3 Gestión de la Seguridad

Para la gestión de la seguridad ante intrusión (véase la Figura A-7) se utilizan los mismos detectores de presencia PIR3-1 y PIR3-2 (y sus correspondientes controladores de entrada) empleados para el apagado automático de la iluminación (véase la Figura A-5). De este modo se logra uno de los principios básicos en domótica: la integración y reutilización de dispositivos para diversas funciones. Las llamadas de servicio generadas ante detecciones de presencia son filtradas por la unidad funcional *temporizador* (TM-2) que a su vez, y a través de su servicio de salida *TMPTempOut* activa un indicador sonoro de alarma (BL-1) con un retardo de 30 segundos. Este retardo es el tiempo del que dispone el usuario para desactivar la alarma una vez que se ha detectado su presencia en la sala. La unidad funcional LG-1 funciona como lógica de habilitación (de tipo *AND*) del disparo del sistema de alarma de intrusión. Un segundo temporizador TM-3 establece el tiempo de activación de la sirena (BL-1) en caso de disparo de la alarma de intrusión. El controlador SMS-1 es el encargado de enviar un mensaje de texto comunicando la incidencia al número de teléfono establecido en sus parámetros.

Por otra parte, para la gestión de la seguridad de bienes y personas (véase la Figura A-7) se utilizan los detectores de inundación WD-1 y WD-2 y el detector de incendio

FD-1, con sus correspondientes unidades funcionales de entrada. Mediante el temporizador TM-4 se activa la sirena BL-2 que genera un aviso acústico en caso de incidencia. Asimismo, el controlador SMS-2 envía un mensaje de texto comunicando la alarma al número de teléfono establecido en sus parámetros

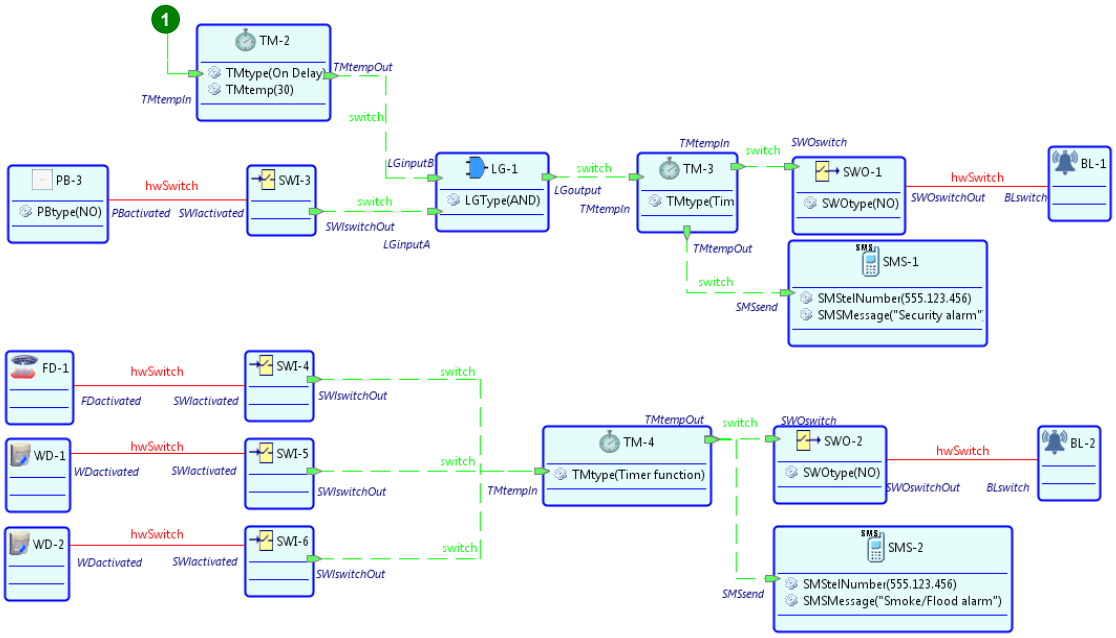


Figura A-7. Modelo de seguridad con el DSL.

A.5.4 Gestión de la Climatización

La Figura A-8 ilustra el modelo del DSL para la gestión de la climatización. La unidad funcional de tipo *controlador de temperatura* (TC-1) realiza la gestión del clima a través de su conjunto de servicios. Los pulsadores *Temp+* y *Temp-*, generan, a través de sus unidades funcionales de entrada (SWI-6 y SWI-7) las llamadas a los servicios *TCsetPoint+* y *TCSetPoint-* de TC-1 para permitir al usuario el incremento / decremento de la temperatura de consigna (deseada) de la estancia en pasos de 1˚C. Mediante el *sensor de temperatura* TS-1 se mide la temperatura actual, que se envía al *controlador de clima* (TC-1) mediante la unidad funcional de entrada analógica AI-1. Además, mediante el servicio *TCcomfortStdby*, conectado al temporizador TM-1 de los detectores de presencia, el *controlador de temperatura* pasa automáticamente a modo de bajo consumo (*standby*) cuando no se detecta presencia en la sala. Para ello se reduce o incrementa (si se está en modo calor o frío) la temperatura en el número de grados fijado en el parámetro *TCstdbytempVariation*. El *programador semanal* WT-1 hace las llamadas necesarias al servicio *TCnight* del *controlador de temperatura* (TC-1) para que éste pase a modo noche a las 00:00 horas y salga de dicho modo a las 8:00 de la mañana (horarios y días fijados en lo parámetros de WT-1). El incremento o decremento de la temperatura para el nodo noche se fija en el parámetro *TCnightTempVariation* de TC-1.

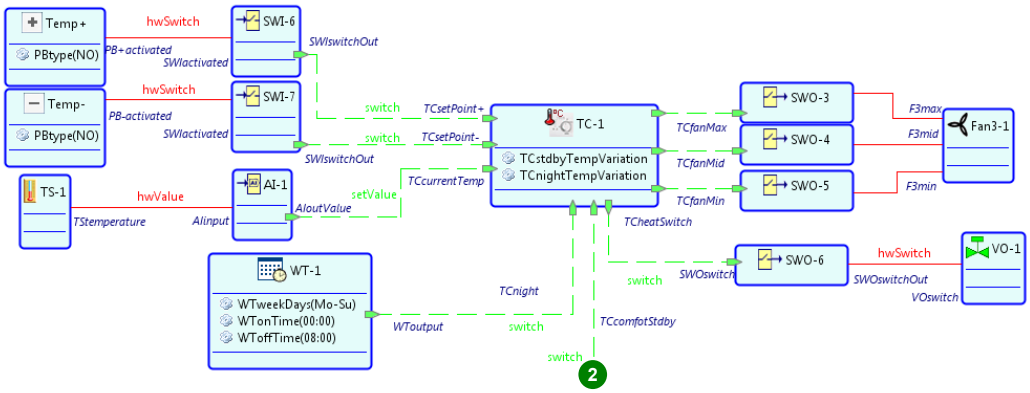


Figura A-8. Modelo de climatización con el DSL.

A.5.5 Escenas

Se han modelado dos escenas para el uso de la sala de juntas, una para realizar presentaciones y otra para realizar debates. En el modelo de la Figura A-9 se pueden observar ambas escenas. La escena "*Presentation*" se ejecuta al activar el pulsador PB-4, conectado al controlador de entrada *SW-4*, que desencadenará la ejecución del servicio provisto *start* en la unidad funcional de escena. Así, se realizan las acciones contenidas en los sucesivos pasos de la escena (regular la intensidad de luz hasta el 20%, bajar la veneciana y la pantalla de proyección). El funcionamiento de la escena *"Meeting"* es similar, aunque las acciones diferentes.

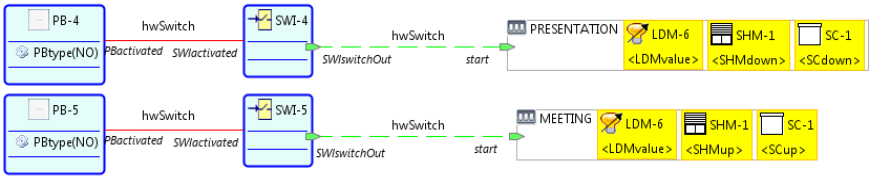


Figura A-9. Modelo de escenas con el DSL.

A N E X O **B**

Ejemplo de Transformaciones entre Niveles CIM - PIM

En este anexo se presentan, de forma detallada, las transformaciones intermedias para obtener el modelo de nivel PIM a partir del modelo de nivel CIM del ejemplo mostrado en el apartado 5.4.1 del capítulo dedicado a la metodología de desarrollo de sistemas domóticos basada en un enfoque dirigido por modelos. De esta manera se puede verificar el correcto funcionamiento de las transformaciones, así como su viabilidad.

B.1 Descripción de las Transformaciones

La Figura B-1 reproduce el modelo inicial de nivel CIM del ejemplo del apartado 5.4.1 en el que enlazan cuatro unidades funcionales: el pulsador PB-1, que es una unidad funcional de tipo final, un elemento de entrada de conmutación SWI-1, uno de salida SWO-1 y una bombilla LO-1, también de tipo final. En la parte superior se muestran las unidades funcionales tal como aparecen en el DSL desarrollado para aplicaciones domóticas, y en la inferior el despliegue de instancias de clases del metamodelo origen (nivel CIM) correspondientes al ejemplo representado con el DSL.

Para facilitar su interpretación, en cada figura de este anexo se han resaltado en verde las clases más importantes implicadas en la transformación (en algunos casos se toman atributos de otras clases, tal como se detalla en las reglas de transformación en el apartado 5.4.1). Sólo se incluyen aquellas transformaciones aplicables al modelo de partida.

En la Figura B-2 se puede observar la creación de los componentes simples *SimpleComponent* a partir de las instancias de unidades funcionales *StdFUInstance*, asociadas a las correspondientes definiciones de unidades funcionales *StdFUDef*. En los atributos de los componentes se incluye la información relativa al nombre de la unidad funcional, su referencia y su tipo.

Una vez creados los componentes, mediante las reglas *2a.RService2PIS* y *2b.IService2PIS* se crean los puertos, interfaces (requeridos o provistos) y los servicios asociados a estos componentes, tal como muestran la Figura B-3 y la Figura B-4 respectivamente.

La Figura B-5 muestra el resultado de aplicar la regla *3.Argument2ServiceParam*, cuya aplicación tiene como consecuencia la creación de los parámetros de los servicios de los componentes. De las subreglas 3a a 3d tan solo es aplicable la regla de transformación *3c.ArgTBool2ServiceParam*, ya que en el modelo CIM sólo hay argumentos de servicio de la clase *BooleanType*. El resultado de la transformación puede apreciarse en la Figura B-6.

Aplicando la regla *4.Param2CompParam* (véase la Figura B-7) se crean los parámetros de componentes *CompParam*, y sus atributos se completan con las reglas *4d, 4e* y *4f* (véase la Figura B-8, que recorren las asociaciones de tipo enumerado de un parámetro y obtienen una cadena de tipo *String* con la lista completa de *Items* y el *Item* seleccionado.

Finalmente, en la Figura B-9 se muestra el resultado de aplicar la regla *5.FUnitLink2PortLink*, que transforma los enlaces entre servicios de unidades funcionales del modelo CIM de origen (FULink) en enlaces entre puertos (PortLink) en el modelo PIM de componentes de destino.

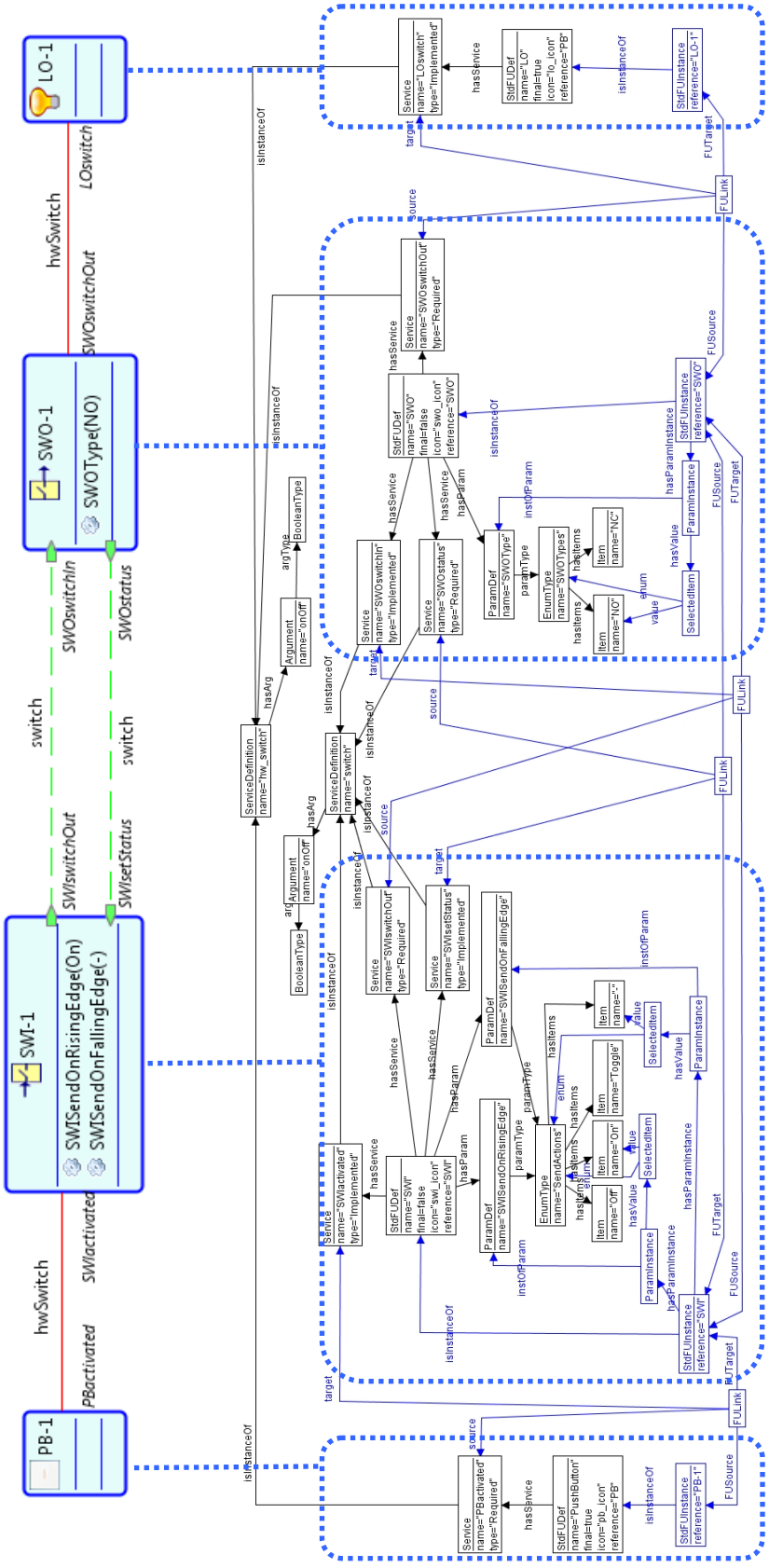


Figura B-1. Modelo inicial de nivel CIM.

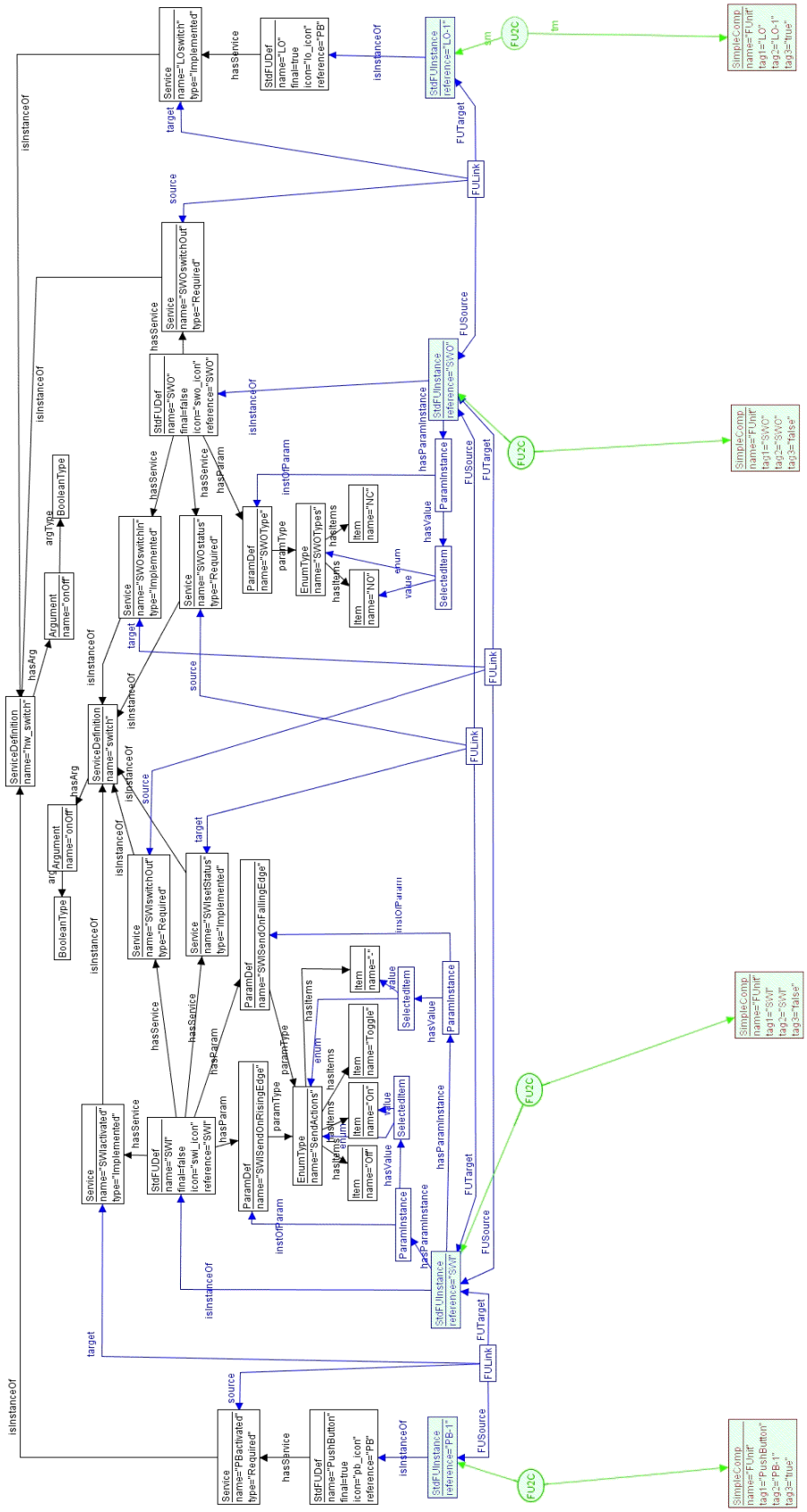


Figura B-2. Modelos tras aplicar la regla *1a.StdFU2Component*.

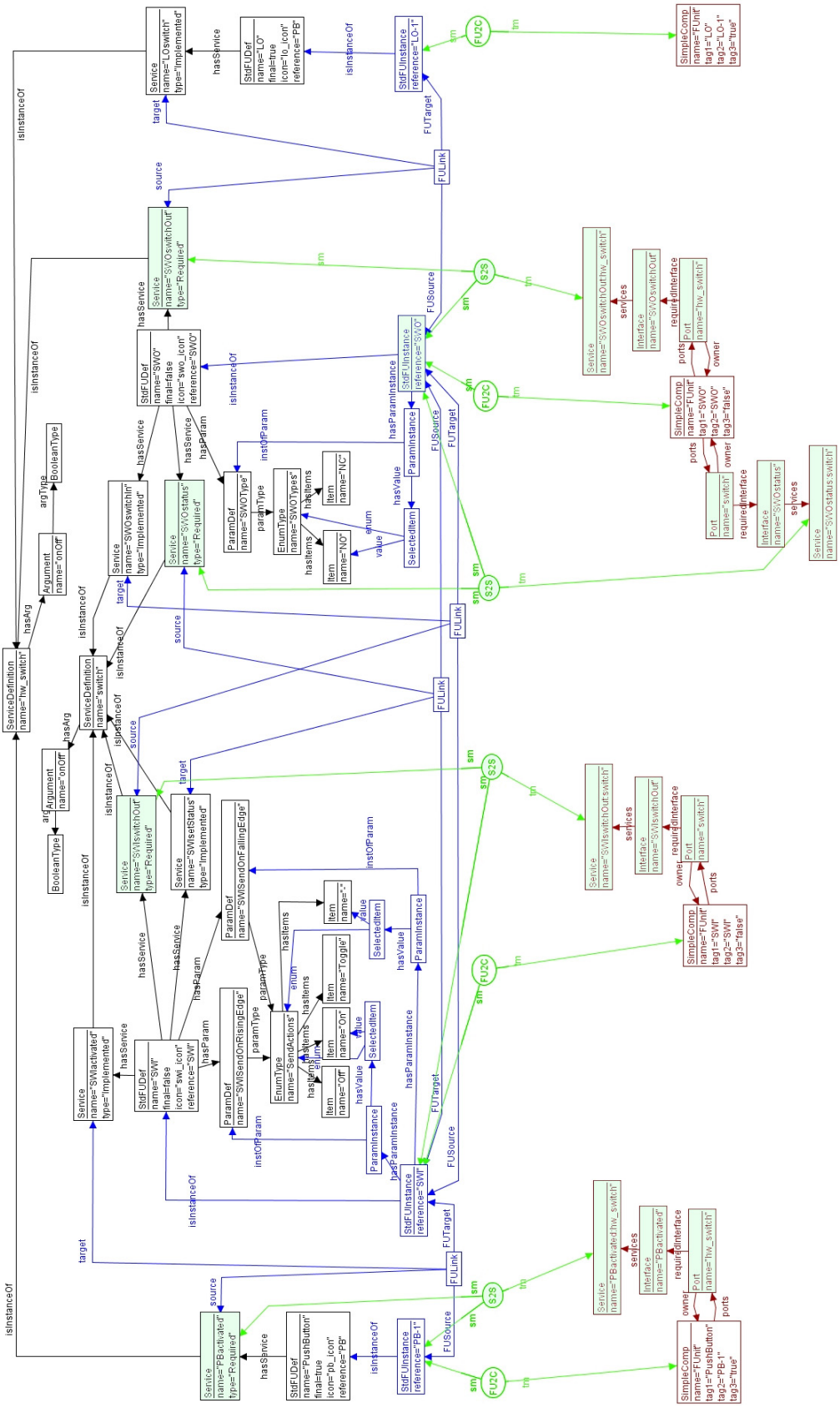


Figura B-3. Modelos tras aplicar la regla: *2a.RService2PIS*.

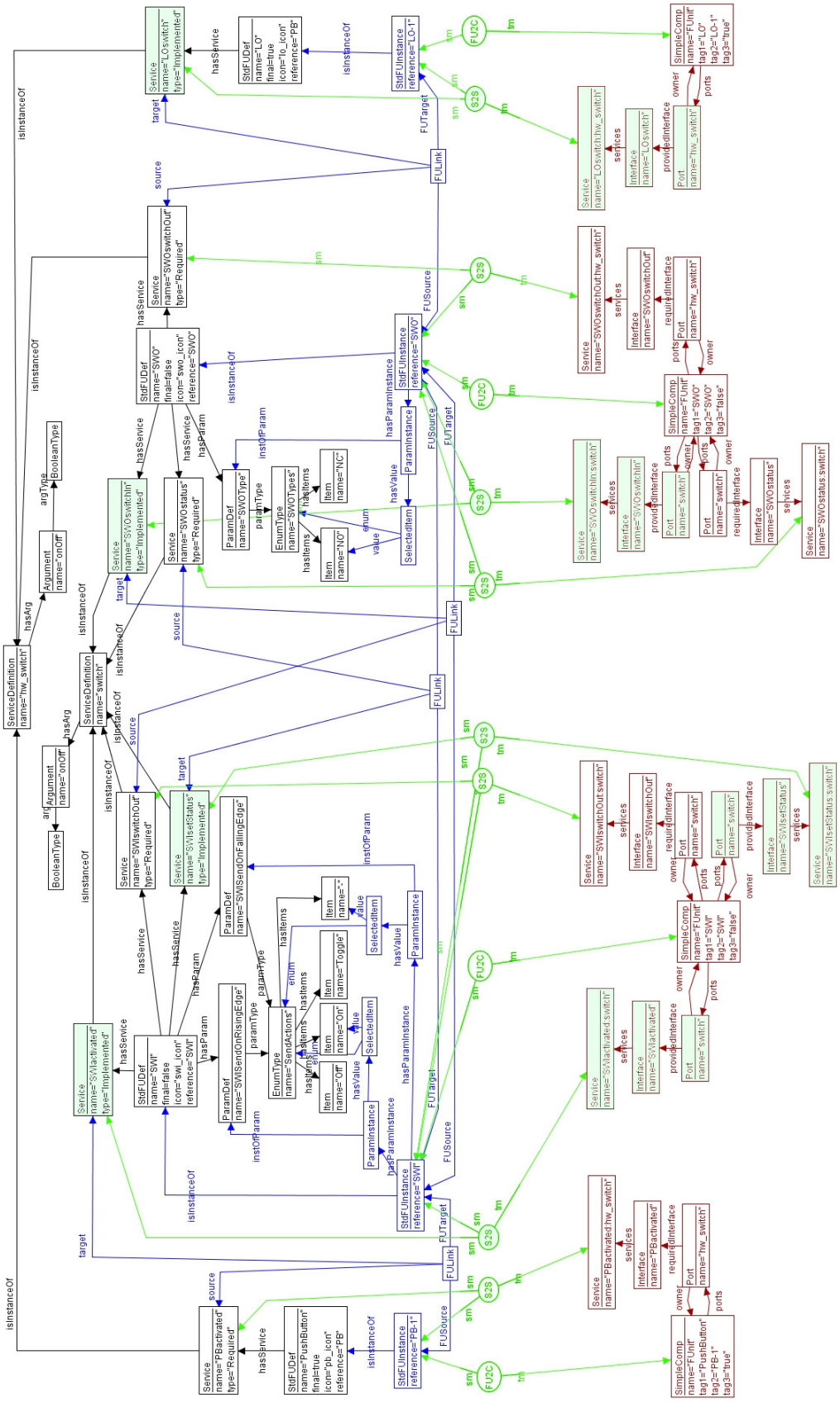


Figura B-4. Modelos tras aplicar la regla *2b.IService2PIS*.

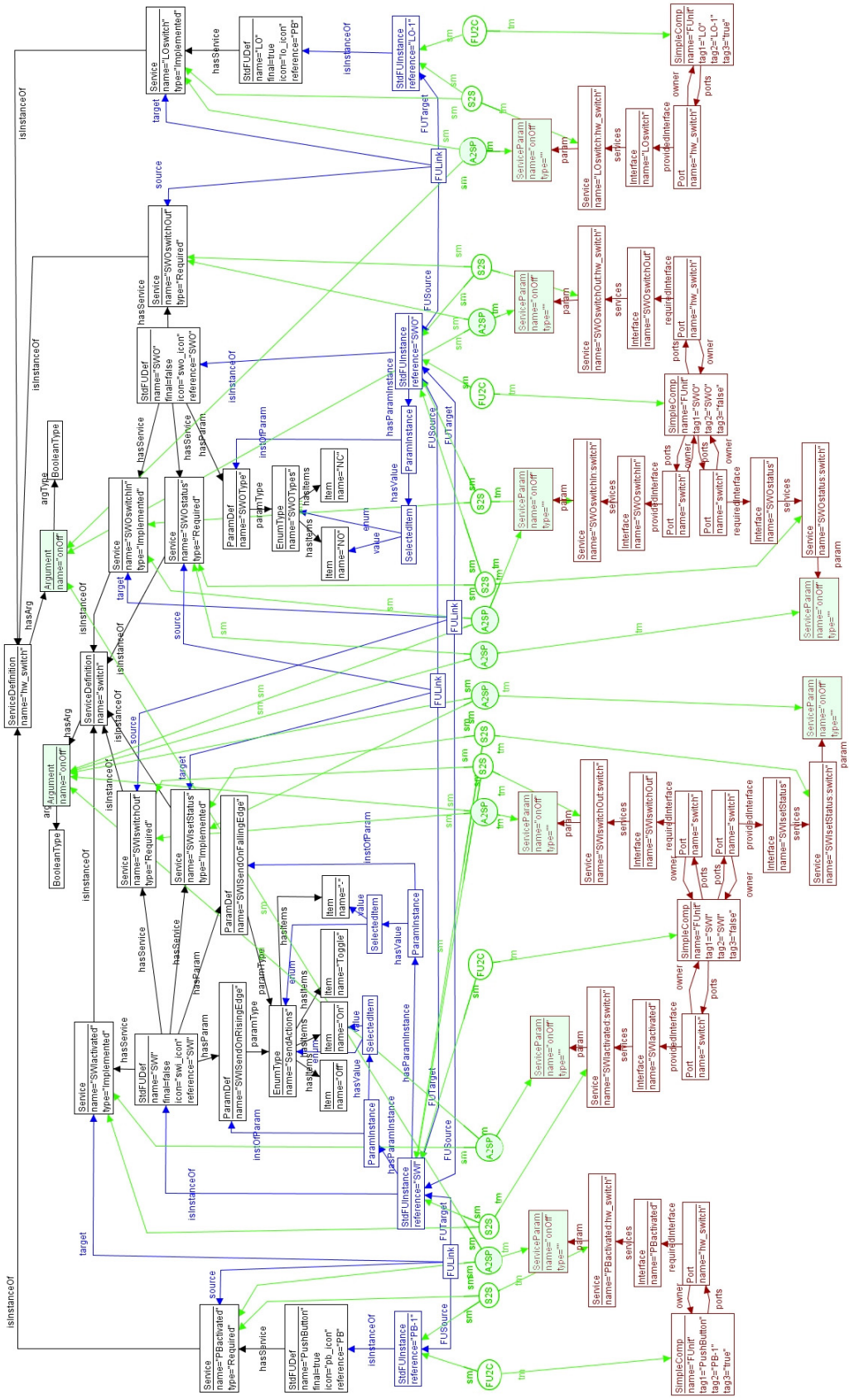


Figura B-5. Modelos tras aplicar la regla *3.Argument2ServiceParam*.

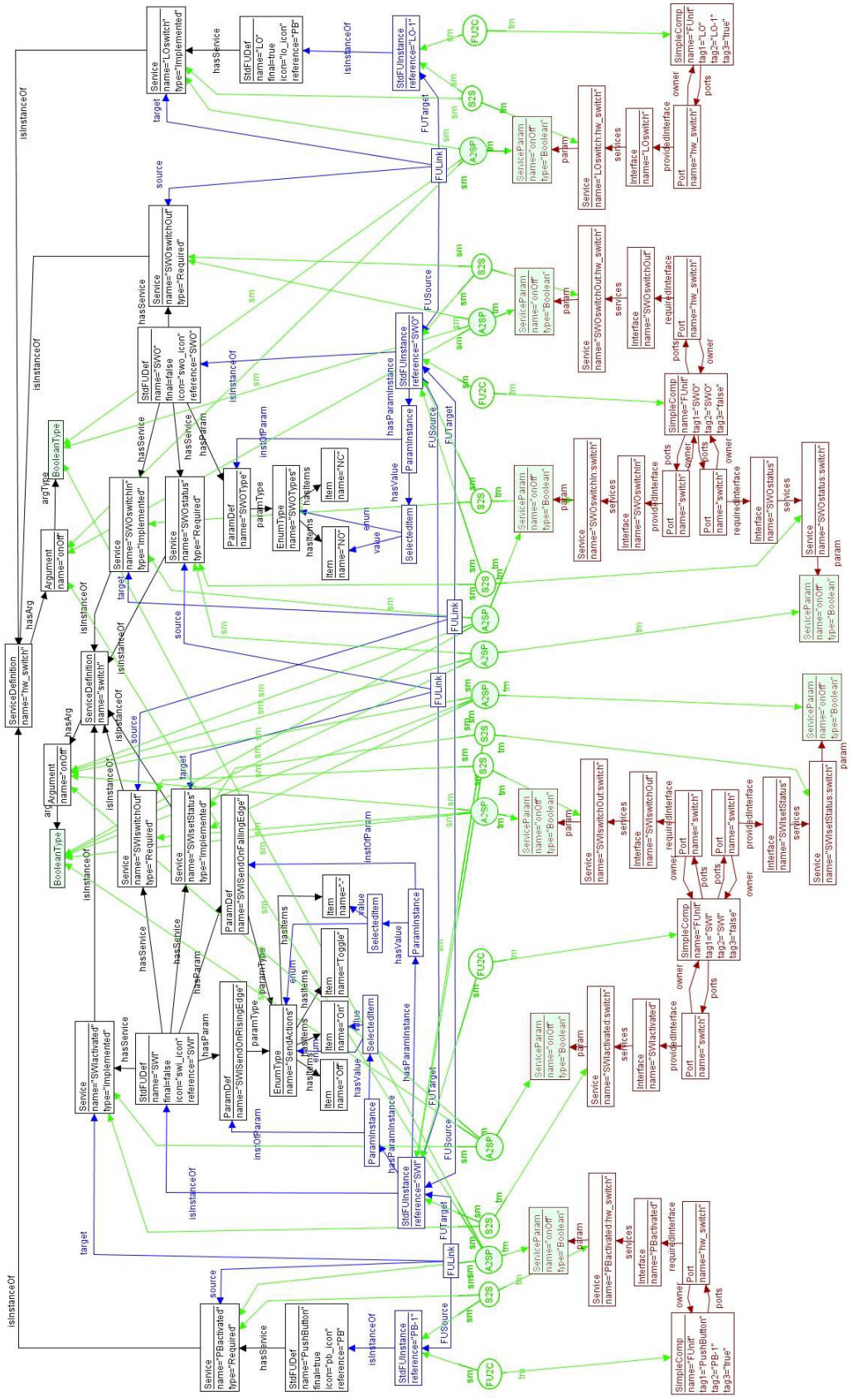


Figura B-6. Modelos tras aplicar la regla *3c.ArgTBool2ServiceParam*.

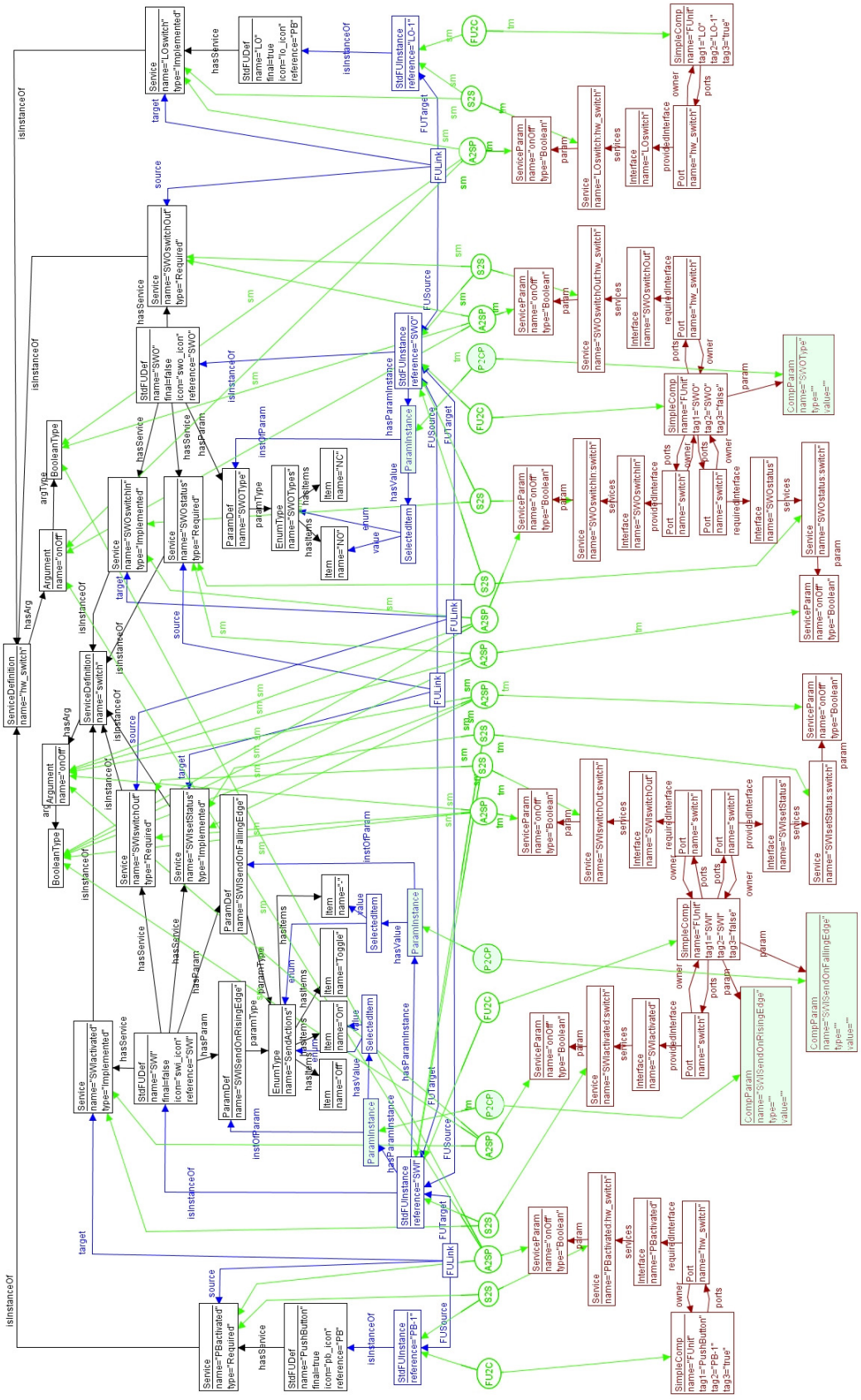


Figura B-7. Modelos tras aplicar la regla *4.Param2CompParam*.

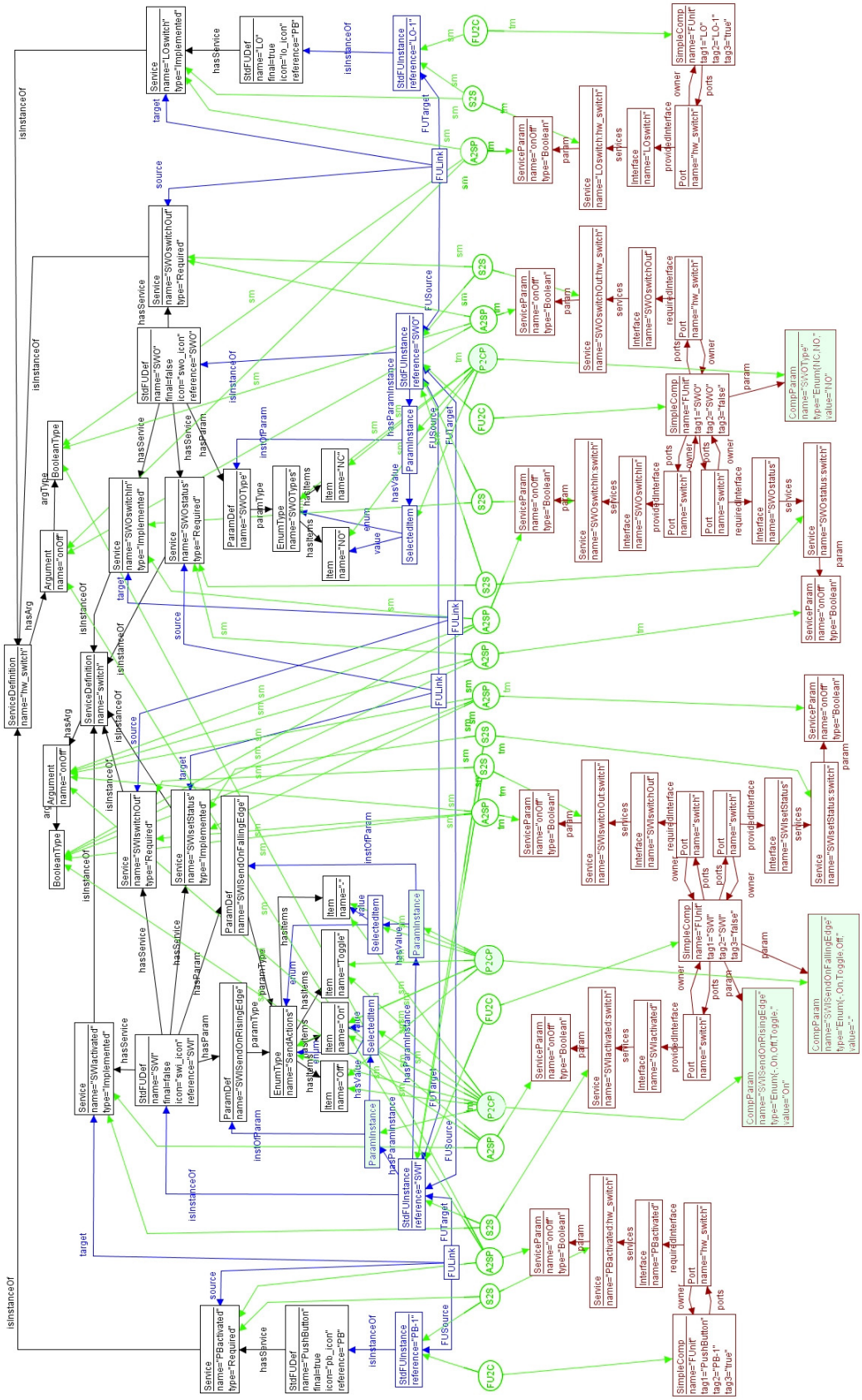


Figura B-8. Modelos tras aplicar las reglas *4d.ParamEnum2CompParam*, *4e.ParamEnumItems2CompParam* y *4f.ParamEnumSelItem2CompParam*.

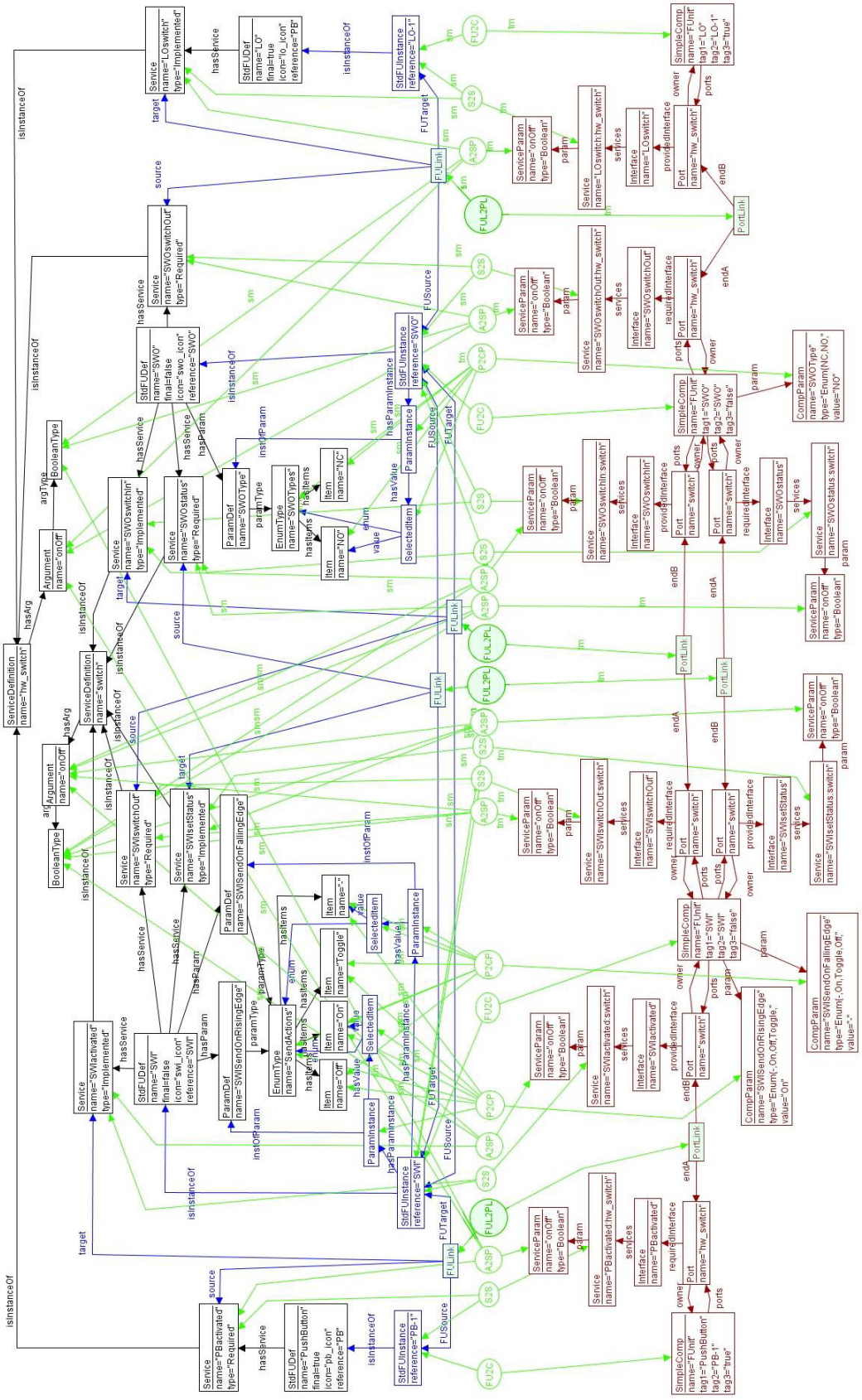


Figura B-9. Modelos tras aplicar la regla *5.FUnitLink2PortLink*.

Bibliografía

[AENOR 08] Asociación Española de Normalización y Certificación. Documentación de normalización. 2008. Disponible en http://www.aenor.es/desarrollo/normalizacion/quees/ventajas.asp

[Acuña 08] Acuña, R.: "Domótica, La Casa Inteligente". 2008. Disponible en http://www2.udec.cl/~racuna/domotica/

[AGG 08] Taentzer, G.: "AGG: The Attributed Graph Grammar System". 2008. Available at http://tfs.cs.tu-berlin.de/agg/

[Agile 01] Beck, K. et al: "Agile Manifesto". 2001. Available at http://www.agilemanifesto.org/

[Alonso 08a] Alonso, D.: "Desarrollo de Software para Robots de Servicio: un enfoque Dirigido por Modelos y Orientado a Componentes". Tesis Doctoral, Universidad Politécnica de Cartagena, 2008.

[Alonso 08b] Alonso, D.; Vicente-Chicote, C.; Barais, O.: "V3Studio: A Component-Based Architecture Modeling Language". Proceedings of the 15th Annual IEEE International Conference and Workshop on the Engineering of Computer Based Systems (ECBS 2008), pp. 346-355, 2008.

[AlSaad 07] Al Saad, M.; Mysliwiec, L.; Schiller, J.: "ScatterPlug: A Plug-in Oriented Framework for Prototyping, Programming and Teaching Wireless Sensor Networks". Proceedings of the Second International Conference on Systems and Networks Communications (ICSNC 2007), IEEE Computer Society, 2007.

[AndroMDA 08] Web del proyecto AndroMDA: Open Source MDA Generator, 2008. Disponible en http://andromda.org/

[Arcstyler 05] Interactive Objects Software: "Arcstyler. The leading platform for Model Driven Architecture (MDA)". 2005. Available at http://www.interactive-objects.com/fileadmin/pdf/products/ArcStyler5_ Whitepaper_ 220205.pdf

[Arregui 97] Arregui, F. J.; Matías, I. R.; López-Amo, M.; Cobo, A.; Echeverría, J.; López-Higuera, J.M.: "Sistema flexible de control de una red de sensores por fibra óptica para un edificio inteligente". Simposio de la Unión Científica Internacional de Radio (URSI-97), vol. 2, pp. 561-564, 1997.

[Astrachan 05] Astrachan, O.; Bruce, K.; Koffman, E.; Kölling, M.; Reges, S.: "Resolved: Objects early has failed". Proceedings of the Thirty-Sixth SIGCSE Technical Symposium on Computer Science Education (SIGCSE 2005), pp. 451-452, 2005.

[Atkinson 01] Atkinson, C. et al: "Component-based Product Line Engineering with UML". Addison-Wesley, 2001.

[ATL 08] ATL: The Atlas Transformation Language Home Page. ATLAS Group (INRIA & LINA), ATL Project, 2006. Available at http://www.sciences.univ-nantes.fr/lina/atl

[Bézivin 01] Bézivin, J.: "From Object-Composition to Model-Transformation with de MDA". Proceedings of the 39th International Conference and Exhibition on Technology of Object-Oriented Languages and Systems (TOOLS39), vol. 39, pp. 350-354, IEEE Computer Society, 2001.

[Bézivin 05a] Bézivin, J.: "On the unification power of models". Journal of Software and Systems Modeling, vol. 42, pp. 171-188, 2005.

[Bézivin 05b] Bézivin, J.; Jouault, F.; Touzet, D.: "Model engineering: from principles to platforms". Chapter from Model Driven Engineering for Distributed Real-Time Embedded Systems, pp. 15-30, Hermes Science Publishing Ltd, 2005.

[Bézivin 05c] Bézivin, J.: "Introduction to Model Engineering. A gentle introduction to a new way of considering the construction and maintenance of information systems". Available at http://www.eclipsecon.com/gmt/omcw/resources/chapter02/downloads/IntroductionToModelEngineering.INRIA.ppt

[Biermann 06] Biermann, E.; Ehrig, K.; Ermel, C.; Köhler, C.; Kuhns, G.; Taentzer, G.: "Tiger EMF Model Transformation Framework (EMT)". 2006. Available at http://tfs.cs.tu-berlin.de/emftrans/papers/userdoc.pdf

[Bruneton 03] Bruneton, E; Coupaye, T.; Stefani, J.: "The Fractal component model". Technical Report Specification V2, The ObjectWeb Consortium, 2003.

[Byoug 00] Byoug-Hee, K.; Kwang-Hyun, C.; Kyoung-Sup, R.K.: "Towards LonWorks technology and its applications to automation". IEEE Proceedings of the 4th Korea-Russia Int'l Symp on Science and Tech., vol. 2, pp. 197-202, 2000.

[Callaway 02] Callaway, E.; Gorday, P.; Hester, L.; Gutierrez, J.A.; Naeve, M.; Heile, B.; Bahl, V.: "Home networking with IEEE 802.15.4: a developing standard for low-rate wireless personal area networks". Communications Magazine, IEEE, vol. 40, issue 8, pp. 70 – 77, 2002.

[CEN 08] European Committee for Standardization home page. 2008. Available at http://ww.cen.eu

[CENELEC 08] European Committee for Electrotechnical Standardization home page. 2008. Available at http://ww.cenelec.eu

[Cetina 07] Cetina, C. et al.: "Tool support for model driven development of pervasive systems". Proceedings of the Fourth International Workshop On Model-Based Methodologies For Pervasive And Embedded Software, pp. 33-41, 2007.

[Clark 08] Clark, T.; Sammut, P.; Willans J.: "Applied Metamodeling. A Foundation for Language Driven Development". CETEVA, 2008. Available at http://www.ceteva.com

[Clements 96] Clements, P.: "A Surrey of achitecture description languages". Proceedings of the 8th International Workshop on Software Specification and Design, ACM, pp. 16-25, 1996

[CWM 03] Object Management Group (OMG), "Common Warehouse Metamodel (CWM) Specification v1.1, formal/2003-03-02". 2003. Available at http://www.omg.org/cgi-bin/apps/doc?formal/03-03-02.pdf

[Czarnecki 03] Czarnecki, K.; Helsen, S.: "Classification of Model Transformation Approaches". Proceedings of the 2nd OOPSLA'03 Workshop on Generative Techniques in the Context of MDA, 2003.

[Czarnecki 05] Czarnecki, K.: "Overview of generative software development". Lecture Notes in Computer Science, vol. 3566 pp.326-341, 2005.

[DeMichiel 06] DeMichiel, L.; Keith, M.:"JSR 220: Enterprise JavaBeans, Version 3.0". 2006. Available at http://java.sun.com/produscts/ejb/docs.html

[Deursen 00] van Deursen A.; Klint P.; Viser J.: "Domain-specific Languages: An Annotated Bibliography". ACM SIGPLAN Notices, vol. 35, issue 6, pp. 26-36, 2000.

[Douligeris 93] C. Douligeris: "Intelligent Home Systems". IEEE Communications Magazine, vol. 5, pp. 52-61, 1993.

[Duc 07] Duc, B.M.: "Real-Time Object Uniform Design Methodology with UML". Springer-Verlag. 2007.

[Eclipse 08] Web oficial de la plataforma Eclipse. 2008. Disponible en http://www.eclipse.org

[EIA 92] Electronic Industries Association: "Draft EIA Home Automation System (CEBus)". Washington DC 1992.

[EN 50065BCD] CENELEC. Norma EN50065-B-C-D: "Asignación de bandas de frecuencias para las redes PLC que se conectan a baja tensión, 2008. Disponible en http://www.cenelec.eu/Cenelec/Code/Frameset.aspx

[Enocean 08] Web oficial de la tecnología Enocean. 2008. Disponible en http://www.enocean.com/

[Estublier 05] Estublier, J.; Vega, G.; Daniela, A.: "Composing Domain-Specific Languages for Wide-scope Software Engineering Applications". Proceedings of the 8th International Conference on Model Driven Engineering Languages and Systems (MoDELS 2005), Lecture Notes on Computer Science, vol. 3713, pp. 69-83, 2005.

[ETSI 08] European Telecommunications Standards Institute home page. 2008. Available at http://www.etsi.com

[Fernández 01] Fernández, C.; Matías, J.R.; Mardones, A.J.; Arregi, F.J.; Jimenez, M.; Vera, J.A.; Roca, J.: "EIB remote control through a secure channel via Internet: Domoware". Proceedings of the European Installation Bus Scientific Conference (EIB 2001), 2001.

[Fujieda 00] Fujieda, H.; Horiike, Y.; Yamamoto, T.; Nomura, T.: "A wireless home network and its application systems". IEEE Transactions on Consumer Electronics, vol. 46, issue 2, pp. 283 – 290, 2000.

[Gamma 95] Gamma, E.; Helm, R.; Johnson, R.; Vlissides J.: "Design patterns: elements of reusable object-oriented software". Addison-Wesley Professional, 1995.

[Gil 05] Gil, B.; Lefebvre, E.: "Maturity of the MDA tool-assisted development process using business archetypes: a case study". Proceedings of the 2005 International Conference of Software Engineering Research and Practice (Serp'05), vol. 1, pp. 375-381, 2005.

[GMF 08] Eclipse Consortium, GMF (Graphical Modeling Framework). Web oficial del proyecto, 2008. Disponible en http://www.eclipse.org/gmf/

[GreAT 08] Web oficial del proyecto GReAT: Graph Rewriting And Transformation, 2006. Disponible en http://www.isis.vanderbilt.edu/Projects/mobies/downloads.asp

[Hakem 02] Hakem, N.; Misson, N.: "Study of the encapsulation of two medium access methods for a wireless home automation network". Proceedings of the 13th IEEE International Symposium on Personal, Indoor and Mobile Radio Communications, vol. 1, pp. 506-510, 2002.

[Hamilton 97] Hamilton, G.: "Java Beans specification 1.101". 1997. Available at http://java.sun.com/products/javabeans/docs/spec.html

[Hernández 97] Hernández, D.: "El futuro económico de la domótica". El Mundo de la Domótica, pp.30, Septiembre 1997.

[Honeywell 02] Web oficial de la tecnología Hometronic, 2008. Disponible en http://www.honeywellsp.com/hw_productos_servicios/hw_residencial/Hw_Control_Residencial.htm

[IEC 08] International Electrotechnical Commission home page. 2008. Available at http://ww.iec.ch

[ISO 08] International Organization for Standardization home page. 2008. Available at http://ww.iso.org

[ITC-BT-51] Ministerio de Industria, Turismo y Comercio, "Instrucción Técnica Complementaria para Baja Tensión: ITC-BT-51 Instalaciones de sistemas de automatización, gestión técnica de la energía y seguridad para viviendas y edificios". Edición febrero de 2007.

[ITU 08] International Telecommunication Union home page. 2008. Available at http://ww.itu.int

[Ivers 02] Ivers, J.; Sinha, N.; Wallnau, K.: "A Basis for Composition Language CL". Technical Report CMU/SEI-2002-TN-026, CMU SEI, 2002.

[JET 07] Eclipse Consortium, Java Emitter Templates (JET). 2007. Available at http://www.eclipse.org/modeling/m2t/?project=jet

[Kang 06] Kang, K.; Kim, J.; Kim, K.; Shin, E. y Huh, M.: "FORM: A feature-oriented reuse method with domain-specific reference architectures". Annals of Software Engineering, vol. 5, issue 0, pp. 143-168, Springer-Verlag, 1998.

[Kastner 05] Kastner, W.; Neugschwandtner, G.; Kogler, M.: "An open approach to EIB/KNX software development". Proceedings of the Sixth International Conference on Fieldbus Systems and their Applications (IFAC 2005), vol. 6, issue 1, 2005.

[Kauffman 00] Kauffmann, P.; Misson, M.: "Study of a home automation network radio variant: the case of Teledomotis". Proceedings of the Fifth IEEE Symposium on Computers and Communications (ISCC 2000), pp. 526-531, 2000.

[Kent 02] Kent, S.: "Model Driven Engineering". Proceedings of the 3rd International Conference on Integrated Formal Methods (IFM 2002), Lecture Notes on Computer Science, vol. 2335, pp. 286-298, Springer-Verlag, 2002.

[Klaus 01] Klaus, D.; Verlang, F.: "Powerline Communications". Upper Saddle River, NJ 07458, Ed. Prentice Hall PTR, 2001.

[Kleppe 03] Kleppe, A.; Warmer, J.; Bast, W.: "MDA Explained: The Model Driven Architecture-Practice and Promise". Addison-Wesley Professional, 2003.

[KNX 04] Konnex Association: "KNX System Architecture". 2004. Available at http://www.knx.org/fileadmin/downloads/03-KNX_Standard/KNX_Standard_Public_Documents/KNX_System_Architecture.pdf

[Kojima 93] Kojima, H.; Lijima, Y.: "Visual Communicacion System in Apartment House Using Fiber Optic". Proceedings of the 3rd IEEE International Conference on Consumer Electronics, pp. 362-364, June 1993.

[Konnex 08] Web oficial de la asociación del estándar Konnex, 2008. Disponible en http://www.konnex.org

[Kyselytsya 06] Kyselytsya, Y.; Weinzierl, T.: "Implementation of the KNX Standard". Proceedings of the Technical KNX Scientific Conference (KNX 2006), Technische Universität Wien, 2006. Available at http://www.wimac-at-home.de/v2/publikationen/

[Lara 02] Lara, J.; Vangheluwe, H.: "ATOM3: A Tool for Multi-Formalism Modelling and Meta-Modelling". Proceedings of Fundamental Approaches to Software Engineering (FASE'02), vol. 2306, pp. 174 – 188, Springer-Verlag, 2002.

[Larman 02] Larman, C.: "Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and the Unified Process". Prentice Hall, 2002.

[Larousse 08] Gran diccionario de la lengua española, Larousse Editorial, 2008.

[Lau 05] Lau, K.K.; Wang, Z.: "A taxonomy of software component models". Proceedings of the 31st Conference on Software Engineering and Advanced Applications (EUROMICRO 2005), pp. 88-95, 2005.

[Lemesle 98] Lemesle, R.: "Transformation rules based on metamodeling". EDOC'98, Proceedings of the Second International Enterprise Distributed Object Computing Workshop (EDOC'98), pp. 113-122, 1998. Available at http://www.sciences.univ-nantes.fr/lina/atl/publications

[Lennox 04] Lennox, J.; Schulzrinne, H.: "CPL: A Language for User Control of Internet Telephony Services". Internet Engineering task Force, IPTel Working Group, 2004.

[Leven 04] Levendovszky, T.; Lengyel, L.; Mezei, G.; Charaf, H.: "A Systematic Approach to Metamodeling Environments and Model Transformation Systems in VMTS". Proceedings of the 2nd International Workshop on Graph Based Tools (GraBaTs'04), Electronic Notes in Theoretical Computer Science, vol. 17, issue 1, pp. 65-73, 2004.

[Libro Blanco 03] Fundación Telefónica, "Libro Blanco del Hogar Digital y las Infraestructuras Comunes de Telecomunicaciones". 2003. Disponible en http://www.telefonica.es/sociedaddelainformacion/html/publicaciones_libroblanco.shtml

[Losilla 07a] Losilla, F. et al.: "Wireless sensor network application development: An architecture-centric MDE approach". Lecture notes in computer science, vol. 4758, pp. 179-194, 2007.

[Losilla 07b] Losilla, F. et al. "A WSAN solution for irrigation control from a model driven perspective". Wireless Sensor And Actor Networks pp. 35-46, 2007.

[Luoma 04] Luoma J.; Nelly S.; Tolvanen J.P.: "Defining Domain-Specific Modeling Languages: Collected Experiences". Proceedings of the 4th OOPSA Workshop on Domain-Specific Visual Languages, 2004.

[Matías 99] Matías, I. R.; M. López-Amo: "Las telecomunicaciones en el siglo XXI. Introducción". Proyectar en Navarra, vol. 47, pp. 138-142, Julio 1999.

[Matula 03] Matula, M.: "NetBeans Metadata Repository". Metadata Repository (MDR) Whitepaper, 2003. Available at http://mdr.netbeans.org/MDR-whitepaper.pdf

[MDA 03] Object Management Group (OMG), "Model Driven Architecture Guide Version v1.0.1, omg/2003-06-01". 2003. Available at http://www.omg.org/docs/omg/03-06-01.pdf

[Mellor 04] Mellor, S.; Scott, K.; Uhl, A.; Weise D.: "MDA Distilled. Object Technology". Addison-Wesley Professional, 1ª edición, 2004.

[Mens 06] Mens, T.; van-Gorp, P.: "A Taxonomy of Model Transformation". Proceedings of the International Workshop on Graph and Model Transformation (GraMoT 2005), Electronic Notes in Theoretical Computer Science, vol. 152, pp. 125-142, Elsevier Science Inc., 2006.

[Mercahome 04] Institut Cerdà; CEDOM (Asociación Española de Domótica); CASADOMO: "Proyecto Mercahome". Programa PROFIT del Ministerio de Industria, Turismo y Comercio, 2004.

[Mernik 01] Mernik, M.; Lämmel, R. (editors): "Special issue on domain-specific languages, Part I". Journal for Computing and Information Technology, vol. 9, issue 4, 2001.

[Mernik 02] Mernik M.; Lämmel R., (editors): "Special issue on domain-specific languages, Part II". Journal for Computing and Information Technology, vol. 10, issue 1, 2002.

[Mernik 05] Mernik M.; Heering J.; Sloane A.: "When and how to develop domain-specific languages". ACM Computer Survey, vol. 37, issue 4, 2005.

[Mint 08] Estudio MINT-CASADOMO 2008: "Sistemas de Domótica y Seguridad en Viviendas de Nueva Promoción". 2008. Disponible en http://www.casadomo.com/noticiasDetalle.aspx?id=10907&c=6

[MOF 04] Object Management Group (OMG), "Meta-Object Facility (MOF) Specification v2.0, ptc/04-10-15". 2004. Disponible en http://www.omg.org/cgi-bin/apps/doc?formal/06-01-01.pdf

[MOFLON 08] Web oficial del proyecto MOFLON, 2008. Disponible en http://www.moflon.org/

[MOF-QVT 05] Object Management Group (OMG), "Meta-Object Facility (MOF) v2.0 Query/View/Transformation Specification, ptc/05-11-01". 2005. Available at http://www.omg.org/cgi-bin/apps/doc?ptc/05-11-01.pdf

[MoTMoT 06] Muliawan, O.: "MoTMoT: a template-based Model to Model Transformer". 2006. Available at en http://www.fots.ua.ac.be/motmot/index.php

[Muñiz 96] Muñiz, C.; López-Amo, M.; López Higuera, J.M.: "Fibra óptica en edificios inteligentes. Integración de servicios". Mundo Electrónico, pp. 47-53, Enero 1996.

[Muñoz 06] Muñoz J.; Pelechano V.; Cetina C.: "Implementing a Pervasive Meetings Room: A Model Driven Approach". Proceedings of the 3rd International Workshop on Ubiquitous Computing (IWUC 2006), pp. 13 – 20, May 2006.

[Muñoz 07] Munoz J.; Pelechano V.; Cetina C.: "Software Engineering for Pervasive Systems. Applying Models, Frameworks and Transformations". Proceedings of the IEEE International Conference on Pervasive Services, pp. 290-294, 2007.

[Nierstrasz 02] Nierstrasz, O. et al.: "A component model for field devices". Proceedings of the 1st Int. IFIP/ACM Working Conference on Component Deployment, pp. 200-209, ACM Press, 2002.

[Nokia 03] Nokia: "Nokia series 60 SDK documentation, version 2.0, 2003. Available at http://forum.nokia.com

[Nozick 88] Nozick, J.: "La maison intelligente". Editions du Moniteur, 1988.

[Nunes 00] Nenes, R.J.C.; Delgado, J.C.M.: "An Internet application for home automation". Proceedings of the 10th IEEE Electrotechnical Conference, vol. 1, pp. 298-301, May 2000.

[OCL 06] Object Management Group (OMG), "Object Constraint Language (OCL) Specification v2.0, formal/06/05/01". 2006. Available at http://www.omg.org/cgi-bin/apps/doc?formal/06-05-01.pdf

[OMG 04] Object Management Group (OMG): Common Object Request Broker Architecture (CORbA/IIOP) formal/04-03-12 Specification. 2004. Available at http://www.omg.org/docs/formal/04-03-12.pdf

[Ommerging 00] Ommering, R.; Linden, F.; Kramer, J.; Magee, J.: "The Koala component model for consumer electronics software". IEEE Computer, vol. 33, issue 3, pp. 78-85, 2000.

[Ortiz 05] Ortiz, F.J.: "Arquitectura de referencia para Unidades de Control de Robots de Servicio Teleoperados". Tesis Doctoral, Dpto. Tecnología Electrónica, Universidad Politécnica de Cartagena, 2005.

[Ortiz 07] Ortiz, F.J. et al.: "Experiences using a component-oriented architectural framework for robots and its improvement with a MDE approach". Lecture Notes in Computer Science, vol. 4758, pp. 335-338, 2007.

[OSGi 08] Open Service Gateway (OSGi) initiative Alliance. Available at http://ww.osgi.org

[Ott 99] Ott, R.; Reiter, H.: "Connecting EIB components to distributed Java applications". Proceedings of the IEEE International Conference on Emerging Technologies and Factory Automation, vol. 1, pp. 23-26, 1999.

[Plasil 98] Plasil, F.; Balek, D.; Janecek, R.: "SOFA/DCUP: Architecture for component trading and dynamic updating". Proceedings of the 4th International Conference on Configurable Distributed Systems (ICCDS '98), pp. 43-52, IEEE Press, 1998.

[Rothenberg 89] Rothenberg, J.: "The Nature of Modeling". Chapter from "Artificial Intelligence, Simulation & Modeling". John Wiley & Sons, pp. 75-92, 1989.

[Ryan 89] Ryan, J.L.: "Home Automation". Electronics & Communication Engineering Journal, vol. 1, issue 4, pp. 185-192, 1989.

[Schmidt 06] Schmidt, D.C.: "Model-Driven Engineering". Computer, vol. 39, issue 2, pp. 25-31, 2006.

[Seidewitz 03] Seidewitz, E.: "What Models Mean". IEEE Software, vol. 20, issue 5, pp. 26-32, IEEE Computer Society, 2003.

[Selic 03] Selic, B.: "The Pragmatics of Model-Driven Development". IEEE Transactions on Software Engineering, vol. 20, issue 5, pp. 19-25, IEEE Computer Society, 2003.

[Sendall 03] Sendall, S.; Kozaczynski, W.: "Model Transformation: The Heart and Soul of Model-Driven software Development". IEEE Software, vol. 20, issue 5, pp. 42-45, IEEE Computer Society, 2003.

[Sendall 03b] Sendall, S.: "Combining Generative and Graph Transformation Techniques for Model Transformation: An Effective Alliance?". Proceedings of the 18th Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA), 2003.

[Shehata 07] Shehata, M.; Eberlein, A.; Fapojuwo, A.O.: "Managing policy interactions in KNX-based smart homes". Proceedings of the International Computer Software and Applications Conference, vol. 2, pp. 367-372, 2007.

[Sommerville 00] Sommerville, I.: "Software Engineering". 6th Edition, Pearson, Addison Wesley, Chapter 1, 2000.

[Spinellis 01] Spinellis, D.: "Notable Design Patterns for Domain Specific Languages". Journal of Systems and Software, vol. 56, issue 1, pp. 91-99, 2001.

[Sprinkle 03] Sprinkle, J.; Agrawal, A.; Levendovszky, T.; Shi, F; Karsai, G.: "Domain model translation using graph transformations". Proceedings of the 10th IEEE International Conference and Workshop on the Engineering of Computer-Based Systems (ECBS'03), pp. 159, 2003.

[Sprinkle 04] Sprinkle J.: "Model-integrated computing". IEEE potentials, vol. 23, issue 1, pp.28-30, 2004.

[Stahl 06] Stahl, T.; Voelter, M.; Czarnecki, K.: "Model-Driven Software Development: Technology, Engineering, Management". Wiley, 1st edition, 2006.

[Steinberg 08] Steinberg, D.; Budinsky, F.; Paternostro, M.; Merks, E.: "EMF: Eclipse Modeling Framework". Addison-Wesley Professional, 2nd edition, 2008.

[Taetzer 00] Taentzer, G.: "AGG: A tool environment for algebraic graph transformation". Lecture Notes in Computer Science, vol. 1779, pp. 481-488, 2000.

[Tidd 94] Tidd, J.: "Home automation market and technology Networks". Whurr Publishers Ltd, 1994.

[Tolvanen 06] Tolvanen, J.P.: "Domain-Specific Modeling: How to Start Defining Your Own Language". 2006. Available at http://www.devx.com/enterprise/Article/30550

[Tratt 05] Tratt, L.: "Model transformations and tool integration". Software and Systems Modeling, vol. 4, issue 2, pp. 112–122, 2005.

[Tsang 03] Tsang, K.F.; Lee, L.T.: "A novel communication protocol for wireless short command". IEEE Transactions on Consumer Electronics, vol. 49, issue 4, pp. 1020-1027, 2003.

[UML v2.1.1 07] Object Management Group (OMG): "Unified Modeling Language (UML) Superstructure Specification v2.1.1, formal/07-02-06". 2007. Available at http://www.omg.org/cgi-bin/apps/doc?formal/07-02-06.pdf

[Vera 00] Vera, J.A.; Jiménez, M.; Roca, J.: "EIB as a key technology for integrating people with disabilities". Proceedings of the European Installation Bus Scientific Conference (EIB 2000), 2000.

[Vera 01] Vera, J. A.; Jimenez, M.; Roca, J.: "Home control system for the social integration of disabled people". Technology and Health Care, vol. 9, issues 1, 2, pp. 200-202. 2001.

[VIATRA 06] Eclipse, VIATRA2 (VIsual Automated model TRAnsformations) framework. Web oficial del proyecto, 2008. Disponible en http://dev.eclipse.org/viewcvs/indextech.cgi/gmt-home/subprojects/VIATRA2/index.html

[Vicente 07] Vicente-Chicote, C.; Alonso, C.; Chauvel, F.: "V(3)studio: A component-based architecture description meta-model - Extensions to model component behaviour variability". Proceedings of the Second International Conference On Software And Data Technologies (ICSOFT 2007), 2007.

[Vizhanyo 04] Vizhanyo, A.; Agrawal, A.; Shi1, F: "Towards generation of efficient transformations". Lecture Notes in Computer Science, vol. 3286, pp. 298-316, Springer-Verlag, 2004.

[Voelter 07] Voelter M., Groher I., "Product line implementation using aspect-oriented and model-driven software development". Proceedings of the 11th International Software Product Line Conference (SPLC 2007), pp. 233-242, 2007.

[Waks 91] Waks, K.P.: "Utility Management Using Home Automation". IEEE Transactions on Consumer Electronics, vol. 87, issue 2, pp. 168-174, 1991.

[Warmer 99] Warmer, J.B.; Kleppe, A.G.: "Object Constraint Language: Precise Modeling with UML". Addison-Wesley, first edition, 1999.

[Wernetges 07] Werntges, H.: "KNXplorer. A Platform-independent KNXnet/IP Tool for Easy Configuration and Control of KNX Installations". KNX Scientific Conference, 2007.

[Werthschult 01] Werthschulte, K.; Schenider, F.: "Linking devices to the European Installation Bus". Proceedings of the 28th IEEE Instrumentation and Measurement Technology Conference, vol. 2, pp. 827-832, 2001.

[Wile 99] Wile, D.S.; Ramming, J.D.(editors): "Special issue on domain-specific languages". IEEE Transactions on Software Engineering, vol. 25, issue 3, 1999.

[X10 08] Web oficial de la tecnología X10, 2008. Disponible en http://www.x10.com/support/technology1.htm

[XMI 05] Object Management Group (OMG), "XML Metadata Interchange (XMI) Specification v2.1, formal/2005-09-01". 2005. Available at http://www.omg.org/cgi-bin/apps/doc?formal/05-09-01.pdf

[Zitzlsperger 07] Zitzlsperger, S.; Simmel, E.: "KNX@HOME. Standard Web-Browser Based Visualization Of Building Controls And Views For KNX". KNX Scientific Conference, 2007. Available at http://sourceforge.net/projects/knxathome/