Un marco integral para el desarrollo de sistemas domóticos

Francisca Rosique
División de Sistemas e Ingeniería Electrónica (DSIE)
Universidad Politécnica de Cartagena
30202 Cartagena
paqui.rosique@upct.es

Pedro Sánchez
División de Sistemas e Ingeniería Electrónica (DSIE)
Universidad Politécnica de Cartagena
30202 Cartagena
pedro.sanchez@upct.es

Manuel Jiménez
División de Sistemas e Ingeniería Electrónica (DSIE)
Universidad Politécnica de Cartagena
30202 Cartagena
manuel.jimenez@upct.es

Bárbara Álvarez
División de Sistemas e Ingeniería Electrónica (DSIE)
Universidad Politécnica de Cartagena
30202 Cartagena
balvarez@upct.es

Resumen

Este artículo presenta un marco integral para el desarrollo de sistemas domóticos que sigue un enfoque dirigido por modelos, y permite a los desarrolladores obtener código ejecutable para distintas plataformas domóticas. La herramienta de desarrollo presentada proporciona un lenguaje específico del dominio domótico que permite modelar gráficamente el sistema, transformándose estos modelos desde las descripciones de alto nivel hasta el código de las plataformas específicas. Las transformaciones han sido definidas con gramáticas de grafos y extendidas con trazabilidad. Además, se ha incrementado la reutilización de los modelos mediante un catálogo de requisitos. Por todo ello, este marco permite desarrollar aplicaciones domóticas con técnicas que mejoran la calidad tanto del proceso como de los modelos obtenidos.

1. Introducción

El avance tan rápido en electrónica, informática y comunicaciones (que conduce a la miniaturización y mejora de los rendimiento de los ordenadores, sensores y redes) ha dado lugar al desarrollo de nuevas tecnologías en el campo de la domótica [3]. Las aplicaciones domóticas integran funciones de confort, ahorro energético, seguridad y comunicaciones. El objetivo principal de un sistema domótico es dotar a las viviendas de un cierto grado de inteligencia que permita mejorar la calidad de vida de sus habitantes. Tareas tales como el encendido y regulación de luces de forma automática, control de la temperatura, corte de agua y gas cuando se detectan fugas o el control de los dispositivos del hogar de forma remota desde el móvil u ordenador con conexión a Internet, son algunas de las aplicaciones típicas del dominio domótico.

Uno de los principales problemas en el desarrollo de sistemas domóticos es el hecho de que no hay un estándar *de facto* para implementar estas aplicaciones. Existen varios estándares y protocolos adoptados por las empresas que lideran el mercado. Algunos ejemplos notables son KNX (estándares ISO/IEC14543-3-X EN50090), Lonworks (estándares ISO/IEC 14908 EN14908, EIA-709-1) y X 10 (estándar conocido internacionalmente y abierto para comunicación entre dispositivos electrónicos). Como se indica en [17], es improbable que se establezca una única tecnología dominante en el campo de la domótica a corto plazo. Además, cada uno de estas tecnologías proporciona su propio software con el que crear las aplicaciones domóticas y programar los dispositivos. Por lo tanto, se hace imprescindible seleccionar una tecnología en particular (plataforma específica) en la etapa de diseño inicial, puesto que las herramientas y dispositivos finales dependen de esta elección. Estos hechos hacen que el desarrollo de aplicaciones domóticas sea totalmente dependiente de la plataforma, siendo muy complicado incrementar el nivel de abstracción y trabajar con conceptos del dominio domótico en lugar de trabajar con elementos de la tecnología.

Este inconveniente puede solventarse adoptando una técnica de desarrollo dirigida por modelos (MDD) [25]. En este enfoque, el código de las aplicaciones puede generarse automáticamente desde modelos independientes de plataforma.

Actas de las JISBD 2010, pp. 87-97, ISBN: 978-84-92812-51-6

Aunque hace unos años que se utiliza MDD en Ingeniería del Software, no se conocen todavía marcos de desarrollo integrales para el desarrollo de sistemas domóticos que cubran todo el proceso y además aprovechen la infraestructura de ejecución que ya proporcionan los fabricantes de dispositivos.

Por todo esto, se hace necesario obtener una herramienta para el desarrollo de sistemas domóticos que permita la generación automática de código para distintas plataformas. En este documento se presenta un marco de desarrollo integral que permite definir sistemas domóticos a diferentes niveles de abstracción y generar el código necesario. Además, gracias a las ventajas del uso de lenguajes específicos de dominio (DSLs) [15], el desarrollador puede trabajar directamente con elementos gráficos y conceptos propios del dominio.

En resumen, este documento contribuye al estado del arte con las siguientes características:

- Un marco que integra un conjunto de herramientas para definir aplicaciones domóticas a diferentes niveles de abstracción.
- Un conjunto de transformaciones de modelos [16] que permiten a los desarrolladores obtener por completo el código ejecutable.
- Trazabilidad [23] entre artefactos software que permite mejorar la calidad tanto del proceso como de los modelos obtenidos.

El resto del documento se estructura de la siguiente manera. La sección 2 está dedicada a introducir los conceptos básicos de la propuesta y los trabajos relacionados. La sección 3 presenta el marco propuesto y también ofrece una visión general de la implementación usando la plataforma Eclipse. En la sección 4 se explica la herramienta desarrollada para la gestión de la trazabilidad y, finalmente, en la sección 5 se exponen las conclusiones y trabajos futuros.

2. La domótica y MDD

Esta sección se compone de dos subsecciones. Una primera subsección donde se introducen los sistemas domóticos, y una segunda subsección donde se presentan los trabajos relacionados con nuestro enfoque propuesto.

2.1 Desarrollo de sistemas domóticos

En la actualidad, los desarrolladores de aplicaciones domóticas utilizan fundamentalmente herramientas software proporcionadas por los fabricantes de dispositivos (en el caso de sistemas propietarios), o por las asociaciones responsables de proporcionar soporte tecnológico (en el caso de los sistemas estándar). Estas herramientas son por lo general dependientes de la plataforma y orientadas a la generación de código para entornos integrados con un bajo nivel de abstracción. Además, la sintaxis concreta que usan no suele ser muy intuitiva, por lo que el usuario requiere de una formación muy especializada y debe trabajar necesariamente en el espacio de la solución.

Todo el proceso de desarrollo de aplicaciones domóticas se lleva a cabo por tanto por un experto en el dominio que recoge los requisitos del cliente para una instalación (elementos que deben integrarse, los servicios requeridos, la selección de una tecnología concreta, etc.) basándose en su propia experiencia. Este experto lleva a cabo el despliegue de los dispositivos y la programación (usando una infraestructura de desarrollo dependiente de la plataforma) de los dispositivos para obtener la funcionalidad deseada.

Con esta manera de trabajar es complicado obtener las características deseadas en un sistema software tales como: interoperabilidad, flexibilidad, reutilización y mejora productividad. Por otra parte, los desarrolladores suelen centrarse en una determinada tecnología, dejando de lado otras plataformas. Esto es consecuencia del tiempo de formación y especialización requerido (un buen desarrollador debe tener alrededor de 100 horas de cursos y varios meses de práctica). Además, las herramientas para implementar los proyectos domóticos son completamente diferentes en cada plataforma, de modo que el aprendizaje de una nueva tecnología implica un aprendizaje partiendo prácticamente de cero.

2.2 Trabajos relacionados

Es posible encontrar trabajos que intentan obtener de una manera integrada un desarrollo de sistemas domóticos utilizando un enfoque MDD. Entre ellos merece la pena destacar las propuestas de J. Muñoz [20], M. Voelter [28] y G. Nain [21] que

describen l a v entajas d e u tilizar u n en foque d e desarrollo di rigido p or m odelos e n s istemas domóticos. El o bjetivo en todos ellos es aumentar el n ivel d e a bstracción, l a p roductividad y l a calidad d el s oftware, ad emás d e m antener l a independencia c on la pl ataforma de implementación.

Estas p ropuestas r epresentan un buen ejemplo de las v entajas q ue o frece el u so d e M DD en el desarrollo de s istemas do móticos, pero a s u v ez, estos t rabajos pr esentan t ambién a lgunos inconvenientes. En p rimer lu gar, J. Muñoz u tiliza la n otación U ML p ara l a cap tura d e r equisitos, notación no m uy i ntuitiva para l os e xpertos e n el dominio domótico. En el trabajo de M. Voelter se necesita co nstruir u n nuevo m eta-modelo, usando la he rramienta T ree E ditor pr oporcionada p or e l plug-in E MF d e Eclipse, cad a v ez q ue s e d esee implementar una nueva ap licación domótica. E n segundo l ugar, e n todas estas p ropuestas l a generación de código está orientada a la obtención de l os d rivers O SGI (Open S ervice Gateway Initiative) p ara u n s ervidor o u na p lataforma middleware y no a l a pr ogramación de l os dispositivos e n sí . P or lo t anto, siempre s erá necesario u n ex perto d e l a p lataforma es pecífica para la programación de los dispositivos. Por otro lado, G . N ain pr esenta E nTiMid c omo u n middleware compuesto por di versas capas: u na capa en cargada d e l a co nexión e ntre l os dispositivos y l a cap a d e s ervicios, u na cap a puente entre los servicios de tecnologías diversas, tales como Universal Plug and Play (UPnP) y l os Devices P rofile f or W eb S ervices (D PWS) [1 0]. Dicho trabajo define E nTiMid c omo u n middleware que soporta v arios m odelos de servicios d e acces o y d escribe có mo es tos artefactos se generan mediante el enfoque MDD.

Como e lemento di ferenciador de t odos e stos trabajos, e n nuestra pr opuesta e l ni vel de abstracción y us abilidad de l os m odelos de requisitos se incrementa con el uso de un DSL que utiliza c onceptos pr opios de l do minio dom ótico. Además, este trabajo p roporciona l a g eneración automática del código para la programación de los dispositivos d omóticos e n l a t ecnología seleccionada. D e es ta manera, no es n ecesario tener co nocimientos es pecíficos d e cad a plataforma, y por l o t anto, no es n ecesaria l a intervención de un experto en la tecnología.

3. Marco de desarrollo para sistemas domóticos

La F igura 1 muestra lo s principales pasos que se siguen para obtener un sistema domótico haciendo uso del marco propuesto. Como puede observarse, los di ferentes ni veles M DD se c orresponden con: requisitos dom óticos, c onceptos e specíficos de l dominio r ecogidos e n e l D SL, ni vel ba sado e n componentes y, por último, código ejecutable para la plataforma específica. Cada modelo está creado u obt enido de a cuerdo c on s u m eta-modelo siguiendo el paradigma MDD.

Las c orrespondencias e ntre l os r equisitos y e l nivel d el D SL s e han e stablecido de f orma manual. De es ta manera, s e cat alogan l as soluciones parciales para cada requisito domótico. Cuando s e c onstruye una n ueva ap licación, el usuario puede consultar este catálogo e identificar qué requisitos va a considerar. De esta manera, se permite l a r eutilización de s oluciones ad optadas con el D SL en el d esarrollo d e ap licaciones anteriores. Esto s e v e facilitado por e l hecho de que l os r equisitos dom óticos e stán bi en estructurados y es relativamente sencillo asociar a cada r equisito un f ragmento d e D SL que de manera genérica capture la funcionalidad deseada. Desde el n ivel d el D SL h asta l a g eneración d e código, las transformaciones se realizan de forma automática. C ada transformación se r efleja en el correspondiente modelo de t razabilidad que posteriormente s erá pr ocesado p or u na herramienta que pr oporciona un c onjunto d e informes de i nterés al usuario (como s e d escribe más adelante en la subsección 4.2).

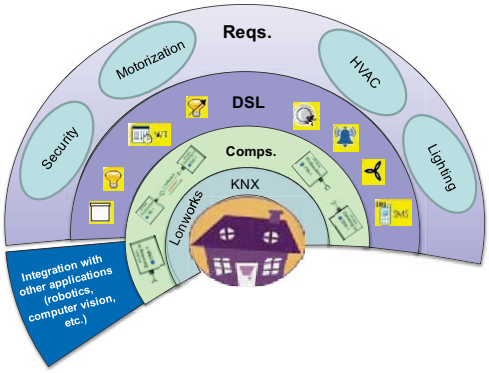

Figura 1: Visión general del marco de desarrollo.

3.1. Gestión de requisitos

Con el fin de integrar la gestión de requisitos en el proceso de desarrollo dirigido por modelos es necesario adoptar un meta-modelo que permita a los desarrolladores catalogar dichos requisitos. En la actualidad es posible encontrar en la literatura diferentes meta-modelos [19][27] con distintas posibilidades a este respecto. Estos meta-modelos mejoran la reutilización de los requisitos y sirven como un modelo de referencia a seguir. En este trabajo, se ha decidido definir un meta-modelo básico con suficiente expresividad para representar los requisitos y las relaciones entre ellos. La Figura 2 muestra el diagrama correspondiente al meta-modelo propuesto.

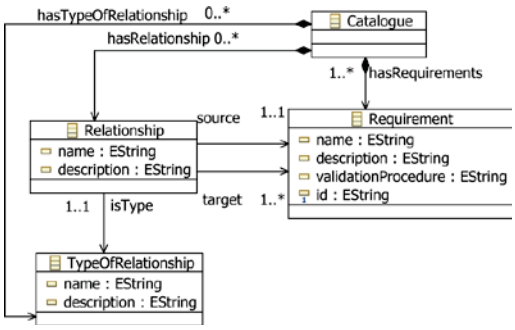

Figura 2: Meta-modelo básico de requisitos.

Este meta-modelo se integra en el entorno de desarrollo y tiene el suficiente nivel de complejidad y versatilidad como para poder adoptar un enfoque de reutilización de requisitos válido para otros ámbitos, y al mismo tiempo, mantener una estructura simple. El elemento raíz se denomina *Catalogue*. En él se recogen los requisitos de la aplicación (*Requirements*). Cada uno de estos requisitos tendrá un nombre, una descripción, un identificador y un procedimiento de pruebas que el usuario deberá comprobar para asegurar que el sistema cumple con las características deseadas. Por ejemplo, dado el requisito "*detección de humo*", el procedimiento de pruebas podría ser "*provocar una situación simulada con humo y comprobar el funcionamiento de las alarmas*". Por otra parte, los requisitos no suelen aparecer aislados sino que están relacionados con otros requisitos. El elemento *Relationship* permite la conexión de un requisito (origen) con uno o varios requisitos destino. El tipo correspondiente a la relación entre requisitos se identifica con el elemento *TypeOfRelationship*.

El usuario puede inspeccionar el catálogo de requisitos y extraer los fragmentos correspondientes a los modelos especificados mediante el DSL. Las correspondencias entre cada requisito y el fragmento asociado del modelo DSL se registran en el modelo de traza. Al proceder de esta manera la especificación de una nueva aplicación en el nivel DSL comienza con la reutilización de fragmentos de modelos. Por último, todos los fragmentos del DSL (reutilizados y nuevos) se integran en un único modelo sobre el cual se realizarán una serie de transformaciones que nos permitirán obtener el código final.

3.2. Descripción del dominio específico

El marco incorpora el uso de un DSL (véase [12] para una descripción detallada de dicho lenguaje) para el modelado de los sistemas domóticos de manera independiente de la plataforma. Este DSL permite describir el sistema de una manera fácil e intuitiva utilizando modelos gráficos.

En cualquier sistema domótico existe una serie de elementos (que denominamos "Unidades Funcionales") que aparecen en todas las tecnologías domóticas. Se diferencian en la arquitectura, protocolos utilizados o módulos disponibles, pero son iguales en cuanto a funcionalidad. Con el fin de promover la

reutilización de estas unidades funcionales y evitar tener que definir varias veces la misma unidad para cada aplicación (incluso varias veces dentro de una misma aplicación), se ha optado por utilizar un Catálogo de unidades funcionales reutilizable, de manera que una vez definido dicho catálogo, éste se pueda utilizar en cualquier aplicación y sólo sea necesario obtener ejemplares de dicho catálogo. Estas unidades funcionales a su vez disponen de unos Servicios gracias a los cuales las unidades podrán interaccionar con otras unidades. Muchos de estos servicios se repiten entre las unidades funcionales, por lo que se ha creado un Catálogo de Servicios con unas Definiciones de Servicios que puedan ser reutilizadas en cualquier unidad funcional.

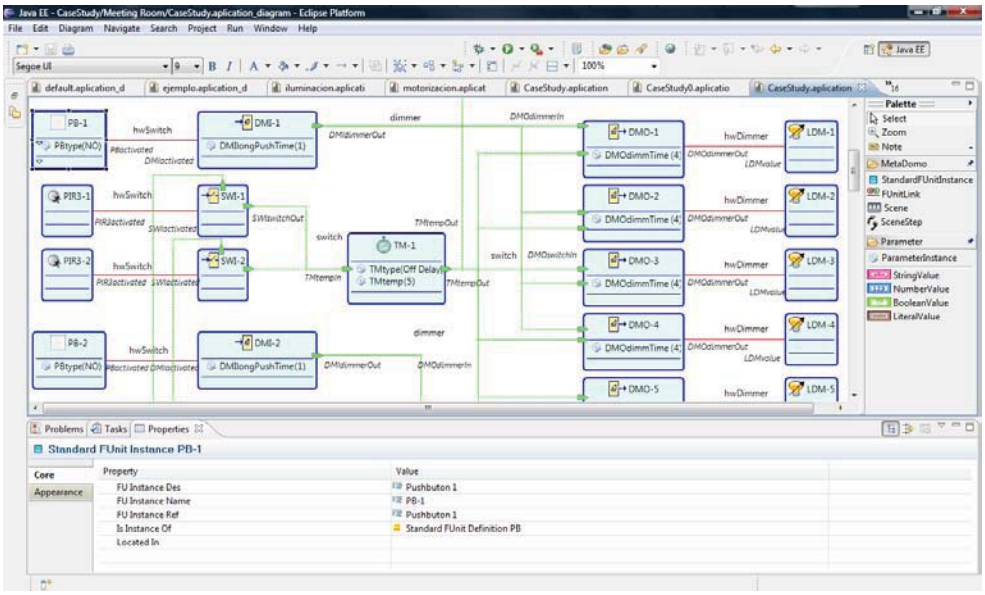

Figura 3: Pantalla ejemplo de sistema modelado usando el DSL para domótica.

Básicamente, una aplicación domótica se define con el DSL mediante la instanciación de las unidades funcionales predefinidas en el catálogo. Estas instancias pueden configurarse dando valores a sus parámetros. Mediante enlaces se puede indicar la forma en la que las unidades funcionales interactúan con el resto del sistema, indicando a su vez los servicios interconectados.

La Figura 3 muestra una captura de pantalla de la herramienta donde se puede observar un modelo implementado con el DSL correspondiente a una aplicación domótica. En particular, se muestra un fragmento relativo a la gestión de iluminación, confort y ahorro energético. Los iconos de las unidades funcionales ayudan a comprender mejor su funcionalidad. La especificación completa viene dada por la vista gráfica del modelo junto con la parametrización de las propiedades correspondientes.

En este ejemplo, se representa el ahorro energético y el confort gracias a la función de apagado automático cuando no se detecta presencia pasados cinco minutos.

Para la implementación del DSL se ha usado la herramienta Graphical Modeling Framework (GMF) que permite generar editores gráficos como plug-in de Eclipse.

3.3. Transformaciones y generación de código

Las transformaciones entre el DSL y la capa inferior basada en componentes han sido definidas usando un enfoque basado en gramáticas de grafos [24], utilizando el plug-in de Eclipse EMT [26]. El hecho de que los modelos se han representado en formato gráfico hace que las transformaciones basadas en gramáticas de grafos sea el enfoque más apropiado ya que son más fáciles de entender y de verificar.

Las transformaciones se expresan mediante reglas que siguen la estructura LHS (Left Hand Side):=RHS (Right Hand Side). En la parte izquierda (LHS) se define un (sub)grafo patrón y en la parte derecha (RHS) se define el (sub)grafo de sustitución. También se puede dar el caso de necesitar una condición de no aplicación, para esto se tiene una tercera parte NAC (Negative Application Condition). Cada vez que se encuentra una correspondencia con el subgrafo patrón (LHS) en el modelo, y no se cumple la condición de no aplicación (NAC), se sustituye dicho patrón por el grafo de sustitución (RHS) [6].

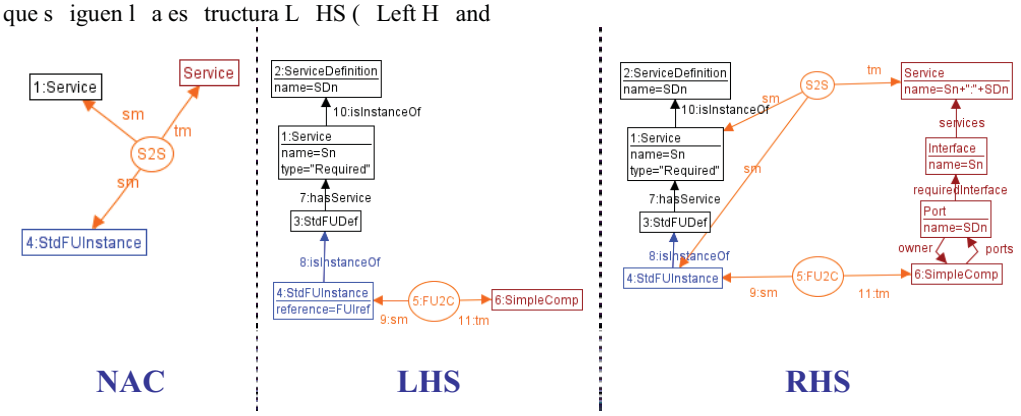

Figura 4: Ejemplo de regla de transformación de grafos: de DSL a modelo de componentes. **Negro:** vista catálogo (DSL); **Rojo:** instancia destino (modelo de componentes); **Azul:** vista aplicación (DSL);**Naranja:** instancia de transformación.

La Figura 4 muestra un ejemplo de una de las reglas de transformación de grafos utilizadas. La regla establece que cada servicio (LHS) existente en el modelo origen debe ser sustituido por un puerto, una interfaz y un servicio del modelo de componentes destino (RHS). Sin embargo, esta regla no se aplica si la transformación ya se ha realizado (NAC). El resultado final de la aplicación de todas las reglas de transformación es un modelo de componentes. Para una completa descripción de las reglas de transformación entre el modelo DSL y el modelo de componentes y los meta-modelos considerados véase [12].

Por medio de transformaciones adicionales, el modelo de componentes es transformado en un modelo para las plataformas específicas seleccionadas. Éste es el último paso del marco de desarrollo con la generación del código ejecutable para la plataforma seleccionada. En este trabajo se ha elegido como primera plataforma de infraestructura especifica la tecnología KNX/EIB, dado que es una tecnología respaldada por estándares internacionales y que es una de las tecnologías líderes en Europa. Para poder realizar este último paso ha sido necesario definir un meta-modelo para la tecnología KNX/EIB basándose en el modelo de objetos de dominio (DOM) que utiliza la herramienta comercial ETS (Engineering Tool Software). Estos modelos específicos de la plataforma son totalmente independientes de las herramientas comerciales y sirven como modelo origen para la transformación final a texto (código). La herramienta JET [5] y el editor de macros ITTools [9] han sido elegidos para la aplicación de estas transformaciones. El código final obtenido viene descrito en el lenguaje de programación VBScript que permite definir las

macros ejecutables en el plug-in ITTools de la herramienta ETS. De esta manera se promueve la reutilización de las herramientas específicas de las plataformas que proporcionan los fabricantes de dispositivos.

Este enfoque es interesante en la medida que se aprovechan las herramientas disponibles en cada tecnología para la ejecución del código, en lugar de construir un nuevo conjunto de herramientas a partir de cero. El punto de partida para un desarrollador tradicional sería la creación del proyecto mediante la iteración manual con la herramienta ETS. Con el enfoque aquí propuesto esa iteración desaparece y la herramienta del fabricante es s implemente un soporte para la ejecución del código obtenido.

4. Trazabilidad entre modelos

En el contexto del desarrollo dirigido por modelos, las técnicas de transformación automática proporcionan la capacidad de trazabilidad [2][4][7]. Debido al uso extendido de las transformaciones (creación automática de artefactos) en el enfoque dirigido por modelos, es fundamental poder saber cómo y por qué se creó un artefacto [1]. La trazabilidad de artefactos software ofrece una información detallada sobre el sistema desarrollado, así como las implicaciones que cualquier cambio pueda tener. Los objetivos que persigue la trazabilidad en el desarrollo de software se pueden resumir en los siguientes puntos [13]:

1. Validar que los requisitos se han tenido en cuenta.
2. Validar que la aplicación cumple con los requisitos obtenidos y si se han satisfecho.
3. Identificar el impacto que puede tener modificar un artefacto software.

Para guardar y después poder procesar los enlaces de trazabilidad es necesario contar con un repositorio de trazas generadas entre los diferentes artefactos software. En [13] se presentan dos enfoques para afrontar la trazabilidad en entornos basados en modelos. Uno de ellos propone integrar la información de trazabilidad en el propio modelo, como por ejemplo, nuevos elementos de modelo tipo estereotipos o atributos. El otro enfoque propone lo contrario, mantener la información en un modelo externo. Este enfoque tiene la ventaja de preservar los modelos limpios (sin información adicional de trazabilidad) y posteriormente realizar los enlaces de trazabilidad entre los modelos. En el trabajo que aquí se presenta se ha considerado esta segunda opción, para ello se ha creado un meta-modelo dedicado a trazabilidad que se describe a continuación.

4.1. Un meta-modelo para trazabilidad

En la literatura existen numerosas propuestas de meta-modelos para dar soporte a trazabilidad [8][13]. Tomando estas propuestas como base, se ha creado un meta-modelo sencillo que aporta la suficiente expresividad como para representar cualquier enlace entre diferentes modelos. Este meta-modelo (véase la figura 5) tiene un elemento llamado *Link* que permite enlazar elementos de cualquier modelo de diferentes meta-modelos, gracias a que tanto la referencia origen y destino es del tipo *ModelElement*. El elemento *CompositeLink* permite definir una jerarquía de enlaces, añadiendo la granularidad necesaria para poder organizar las trazas. Por último, el atributo *linkType* permite a los desarrolladores clasificar las relaciones existentes y distinguir en qué nivel del proceso de desarrollo se ha generado la traza.

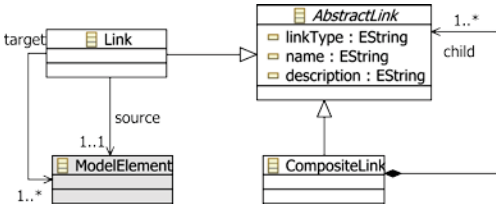

Figura 5: Meta-modelo de trazabilidad.

Los modelos de trazabilidad pueden crearse de forma manual o automática durante el proceso de transformación de modelos [29]. En el presente marco de desarrollo las trazas entre los requisitos domóticos y el modelo creado con el DSL son definidos de forma manual por el usuario. Los elementos que pueden ser trazados en cada dupla origen – destino son los correspondientes a los meta-modelos utilizados (que no se incluyen por requerir un espacio extra para su justificación pero

pueden consultarse en [12]). En las etapas que van desde el DSL hasta la generación de código, las trazas se obtienen automáticamente durante el proceso de transformación modelo a modelo y modelo a texto. Para conseguir esto se han ampliado las reglas de transformación con el fin de poder guardar la información de las trazas en un modelo conforme al meta-modelo presentado en la Figura 5.

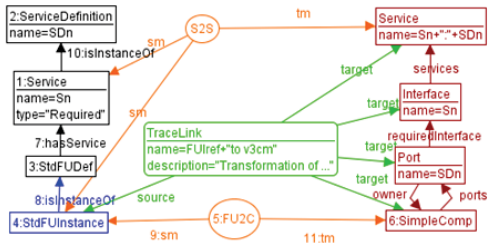

Figura 6: Regla de transformación ampliada con trazabilidad (elementos en color verde).

En la Figura 6 se puede observar la ampliación realizada sobre el grafo de la Figura 4 incluyendo trazabilidad (para simplificar, en este ejemplo tan sólo ha extendido la parte RHS). En dicha figura se observa cómo el nuevo elemento correspondiente a la metamodelo de trazabilidad (elemento representado en color verde) aparece en la regla. De este modo cada elemento *TraceLink* de la regla de transformación recogerá la información necesaria (nombre, descripción, elementos origen y destino) para completar las instancias tipo Link del meta-modelo de trazabilidad

4.2. Una herramienta de gestión de trazabilidad

Varios autores dan recomendaciones sobre las propiedades que se deben tener en cuenta a la hora de diseñar una herramienta de gestión de trazabilidad [8][14], tales como (1) ser capaz de gestionar modelos en diferentes niveles de abstracción, (2) considerar el mismo meta-modelo para todos los niveles de abstracción, (3) almacenar las trazas en un medio persistente, (4) ser capaz de identificar cuándo se creó una traza y la localización de sus elementos de referencia, (5) ser capaz de integrarla con aplicaciones externas, y (6) ser capaz de generar las trazas de manera manual y automática. Teniendo presentes todas estas recomendaciones, se ha desarrollado una herramienta integrada en el marco anterior que permite a los desarrolladores generar informes detallados de las trazas. Estos informes facilitan a los desarrolladores de sistemas dinámicos analizar, por ejemplo, cómo un requisito se ha tenido en cuenta durante el proceso de generación automática de código, así como obtener información de la solución final adoptada.

Esta primera versión de la herramienta se ha desarrollado usando HTML por su portabilidad y por la facilidad que ofrece para incorporar datos XML. A medio plazo se plantea la posibilidad de utilizar herramientas que estén integradas en la plataforma Eclipse (por ejemplo Modelink [18]).

La Figura 7 muestra una captura de pantalla de la herramienta para la gestión de trazabilidad. Se puede ver cómo, para cada una de las trazas, se detalla el nombre y la descripción, así como la información relativa a los elementos origen y destino trazados. La primera fila del informe muestra una traza del tipo *CompositeLink* que organiza toda la información relacionada con los enlaces que estén involucrados en la solución implementada con el DSL para el requisito de la iluminación. En este caso, los enlaces representan las trazas entre los requisitos y los elementos del DSL

El informe de trazabilidad permite al usuario distinguir qué requisitos se han tenido en cuenta, qué elementos arquitectónicos dan soporte a cada uno de ellos, y por último, permite saber qué dispositivos físicos del aplataforma hacen cumplir estos requisitos.

4.3. Beneficios derivados de la trazabilidad

En este apartado, se examinan los principales beneficios que se obtienen con los informes de trazabilidad mencionados anteriormente. Estos beneficios se derivan de la realización las siguientes tareas:

1. *Validar si todos los requisitos han sido tenidos en cuenta.* Se puede comprobar que todos los requisitos están representados por elementos del modelo obtenido con el DSL.
2. *Verificar que el modelo DSL es compatible con los requisitos domóticos.* Los elementos del DSL deben ser validados para comprobar si concuerdan con la semántica de los requisitos oportunos.
3. *Establecer el impacto de una modificación en un requisito domótico.* Es habitual que los usuarios del sistema propongan cambios en los requisitos durante el proceso de desarrollo. El informe de trazabilidad ayuda a los analistas a evaluar el impacto de estos cambios antes de aplicarlos.

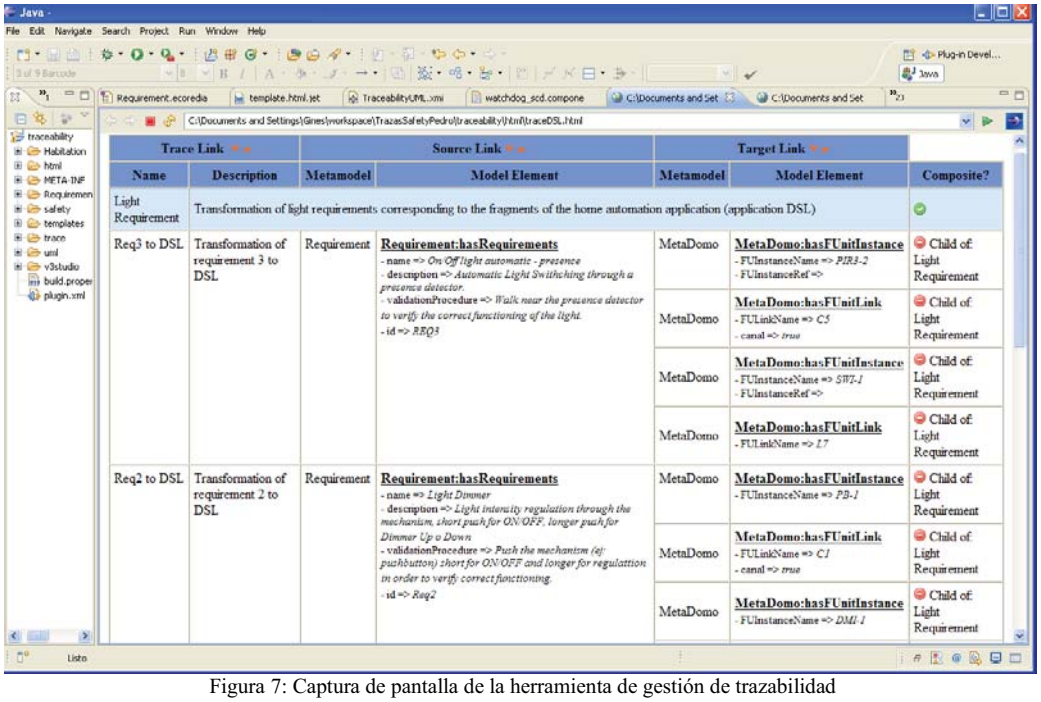

Figura 7: Captura de pantalla de la herramienta de gestión de trazabilidad

5. Conclusiones

En este trabajo se ha presentado un marco de desarrollo para sistemas domóticos siguiendo un enfoque dirigido por modelos. Tradicionalmente, el desarrollo de este tipo de sistemas sólo podía ser realizado por desarrolladores expertos en programación y a la vez en la plataforma de ejecución. El trabajo aquí presentado permite a los desarrolladores con cierta experiencia en el dominio domótico, poder crear la descripción de los sistemas domóticos utilizando un lenguaje específico para este dominio que será transformado automáticamente en código ejecutable.

En la actualidad estamos considerando otras plataformas de ejecución y la interoperabilidad de nuestro marco de desarrollo con otras herramientas (como un animador para validar los modelos antes de la implementación final). Además, se está trabajando en una mejora de la herramienta de gestión de trazabilidad para facilitar, entre otros, un filtrado avanzado de trazas, generación de estadísticas de los artefactos generados en cada nivel, análisis de elementos huérfanos y, por último, la integración de la herramienta en entornos distintos a Eclipse.

Las transformaciones implementadas no funcionan de manera incremental por lo que un cambio en el modelo origen conlleva volver a ejecutar todas las transformaciones. Este es un aspecto susceptible de mejora a medio plazo que se está investigando.

Además, es interesante considerar otras herramientas como SDK BCU desarrollada por la Universidad de Viena [22], que ofrece una forma alternativa para hacer frente a la generación de código para la plataforma KNX/EIB. Esta herramienta es gratuita y puede facilitar la generación de código, y aquí permite escribir código personalizado para dispositivos OEM KNX. De esta manera se puede evitar tener que catalogar la funcionalidad de los dispositivos comerciales.

A fin de validar la integridad de la propuesta se ha planteado como trabajo inmediato un conjunto de experimentos en los que analizar diversos factores, como el tiempo de desarrollo y mantenimiento de las aplicaciones. El factor de usabilidad ha sido probado con resultados muy prometedores como se detalla en [11], en dicho trabajo se presenta con detalle el DSL y algunos ejemplos de uso. Toda esta información será muy valiosa para ampliar y adecuar las versiones futuras de las herramientas desarrolladas a las necesidades de los desarrolladores.

Referencias

[1] Aizenbud-Reshef, N., Nolan, T., Rubin, J., Saham-Gafni, Y.: Model traceability. IBM Syst J., Vol 45 Nº 3, 2006, pp. 515 – 26.

[2] Behrens, T.: Never Without a trace: Practical advice on implementing traceability. Available: http://www.ibm.com/developerworks/rational/library/feb07/behrens.

[3] Chana, M., Campoa, E., Estèvea, D., Fourniols, J.Y.: Smart homes - Current features and future perspectives. Maturitas - Elsevier, Vol. 64 Nº 2, 2009, pp. 90.

[4] Dick, L.: Rich traceability. IEEE Software. Vol. 15 Nº 1, 2002, pp. 95 - 102.

[5] Eclipse Consortium, Java Emitter Templates (JET). Available: http://www.eclipse.org/modeling/m2g/?project=jet.

[6] Fabro, D., Bézivin, J., Valduriez, P.: Weaving models with the eclipse amwplugin. Eclipse Modeling Symposium, Eclipse Summit Europe, 2006.

[7] Gotel, O.: Contribution structures for requirements traceability, Ph.D. Thesis, Imperial College, Department of Computing, London, England, 1995.

[8] Gotel, O., Finkelstein, C.: An analysis of the requirements traceability problem. Proc. of the first Int. Conference on requirements engineering (RE'94), 1994, pp. 94 – 101.

[9] ITTools. Available: http://www.it-gmbh.de/en/products /ittools.htm.

[10] Jammes, F., Mensch, A., Smit H.: Service-oriented device communications using the devices profile for web services. Proc. of the 3rd Int. workshop on Middleware for pervasive and ad-hoc computing (MPAC '05), 2005, pp. 1 – 8.

[11] Jimenez, M., Rosique, F., Sánchez, P., Álvarez, B., Iborra, A.: Habitation: A Domain-Specific Language for Home Automation, IEEE Software, Vol. 26 Nº 4, 2009, pp. 33 - 38.

[12] Jiménez, M.: Development of Home Automation Applications using a Model Driven Approach, PhD Thesis, Technical University of Cartagena, Spain, March 2009; Available: http://hdl.handle.net/10317/846.

[13] Kolovos, D., Paige, R., Polack, F.: On-demand merging of traceability links with models. Proc. of the 2nd EC-MDA Workshop on Traceability, 2006.

[14] Melby, S.: Traceability in Model Driven Engineering. Master Thesis. University Of Oslo, Norway, Nov. 2007 Available: http://urn.nb.no/URN:NBN:no-18721

[15] Mernik, M., Heering, J., Sloane, A.M.: When and How to Develop Domain-Specific Languages, ACM Computing Surveys. Vol. 37 Nº 4, 2005, pp. 316 – 344.

[16] Mens, T., VanGorp, P.: A taxonomy of model transformation, LNCS 152, 2006, pp. 125 – 142.

[17] Miori, V., Tarrini, L., Manca, M.; Tolomei, G.: An open standard solution for domotic interoperability. Consumer Electronics, IEEE Transactions. Vol. 52 Nº 1, 2006, pp. 97 - 103.

[18] Modelink. Available : http://www.eclipse.org/gmt/epsilon/doc/modelink/.

[19] Molina, F., Toval, A .: Integrating u sability requirements t hat can b e ev aluated i n design time in to M odel D riven E ngineering of W eb Information S ystems. Advances i n Engineering S oftware. Elsevier. Vol. 40 Nº 12, 2009, pp.1306 - 1317.

[20] Muñoz, J., P elechano, V., Cetina, C.: Implementing a P ervasive M eetings R oom: A Model D riven A pproach. Proc. of t he 3 rd Int. Workshop on U biquitous C omputing (IWUC 2006), 2006, pp.13 - 20.

[21] Nain, G., Daubert, E., Barais, O., Jézéquel, J.: Using mde t o bui ld a s chizofrenic middleware for hom e/building a utomation. ServiceWave'08: N etworked E uropean Software & S ervices I nitiative (NESSI) Conference, LNCS 5377, 2008, pp. 49 - 61.

[22] Neugschwandtner, G., Kastner, W ., Kogler, M.: Programming fieldbus no des: a R AD approach t o c ustomizable a pplications. 10 th IEEE C onference o n Emerging T echnologies and F actory Automation (ETFA'05), Vol.1, 2005, pp. 1061 - 1064.

[23] Ramesh, B ., J arke, M .: T oward R eference Models f or R equirements T raceability. I EEE Transactions on s oftware e ngineering, V ol.27 Nª 1, 2001, pp. 58 - 93.

[24] Rozenberg, G. Handbook of G raph Grammars a nd C omputing by Graph Transformation, World Scientific, 1997.

[25] Selic, B. The Pragmatics o f M odel-Driven Development, I EEE S oftware Vol. 20 Nº 5 , 2003, pp. 46 – 51.

[26] Tiger D eveloper T eam. Tiger E MF Transformation. A vailable: http://tfs.cs.tu-berlin.de/emftrans.

[27] Vicente-Chicote, C., Moros, B., Toval, A.: Remm-studio: a n i ntegrated m odel-driven environment f or r equirements specification, validation a nd f ormatting, J ournal of O bject Technology. Vol.6 Nº 9, 2007, pp. 437 – 454.

[28] Voelter, M., Groher, I. Product l ine implementation us ing a spect-oriented a nd model-driven s oftware de velopment. Proc. of the 11t h I nt. Software P roduct L ine Conference (SPLC 2 007), 2 007, pp. 233 - 242.

[29] Walderhaug, S., J ohansen, U . Stav, E ., Aagedal, J . Towards a generic s olution f or traceability in MDD. Proc. of the 3rd ECMDA traceability workshop (ECMDA-TW). 2008.