UNIVERSIDAD POLITÉCNICA DE CARTAGENA
Escuela Técnica Superior de Ingeniería de Telecomunicación

**DESARROLLO INTEGRAL DE APLICACIONES
DOMÓTICAS: UNA PERSPECTIVA
METODOLÓGICA**

Tesis Doctoral

Ma^{ra} Francisca Rosique Contreras

Directores:

Dr. D. Pedro Sánchez Palma
Dr. D. Manuel Jiménez Buén^lia

Cartagena, febrero de 2012

Queda autorizada la reproducción integral de esta Tesis,
a efectos de investigación, mediante declaración escrita
del interesado.

Título: Desarrollo integral de aplicaciones domóticas:
una perspectiva metodológica.

Autora: María Francisca Rosique Contreras

Directores: Dr. D. Pedro Sánchez Palma
Dr. D. Manuel Jiménez Buendía

© 2012 M^a Francisca Rosique

El trabajo de Tesis que a continuación se presenta se acoge a la modalidad de Tesis por compendio de publicaciones del Departamento de Tecnologías de la Información y las Comunicaciones de la Universidad Politécnica de Cartagena, de acuerdo con la normativa vigente (reglamento de estudios oficiales de máster y doctorado aprobado en el consejo de gobierno de 13 de abril de 2011). Consta de cuatro artículos, de los cuales, uno de ellos ha sido publicado en la revista “*IEEE Software*”, un segundo en la revista “*Computer Applications in Engineering Education*”, el tercero en la revista “*IEEE Transaction on Computers*” y el último en la revista “*Journal of Systems and Software*”. No obstante, además se enumeran otros trabajos realizados en el marco de esta investigación, que han sido presentados en congresos, otras revistas de menor relevancia e incluso en proceso de elaboración, para su posterior evaluación y, en su caso, publicación.

El nexo común que une estos trabajos es el de profundizar en la generación de un nuevo conocimiento dentro del campo de la ingeniería software, más concretamente en el enfoque de desarrollo dirigido por modelos, siendo su objetivo final, el de mejorar las técnicas actuales de desarrollo de software para sistemas domóticos. Para ello se ha continuado con la línea de investigación iniciada en la Tesis del Dr. D. Manuel Jiménez en el campo de la domótica (Jiménez, 2009), donde se definió un marco general y los elementos iniciales de un lenguaje específico del dominio (DSL) para domótica. Para la Tesis que aquí planteamos se propone por tanto aplicar nuevas técnicas de la Ingeniería del Software que permitan la gestión integral del desarrollo del software domótico en todas sus etapas, abarcando temas muy variados como los lenguajes de modelado, trazabilidad de modelos, ingeniería de requisitos, desarrollo de *framework*, desarrollo de editores y herramientas de soporte, etc.

Las investigaciones planteadas y detalladas en cada uno de los artículos pretenden aportar una contribución parcial a un objetivo tan ambicioso como el planteado.

El siguiente documento se ha dividido en dos partes, una primera parte que permitirá situar al lector en el contexto en el que se ha desarrollado esta Tesis, indicando los objetivos planteados y el estado del arte, en la segunda parte se expondrá el trabajo de Tesis llevado a cabo, para ello se resumirán los resultados alcanzados y aportaciones de los distintos artículos incluidos en esta Tesis por compendio. Los trabajos incluidos en este documento (publicados en ISI-JCR) son:

1. Habitation: A Domain Language for Home Automation (Jimenez et al., 2009), publicado en la revista “*IEEE Software*” (JCR-ISI, IF 2.039, Q1).
2. A tool for facilitating the teaching of smart home application (Jiménez et al. 2011.), publicado en la revista “*Computer Applications in Engineering Education*” (JCR-ISI, IF 0.321, Q3)
3. Introducing Safety Requirements Traceability Support in Model- Driven Development of Robotic Applications (Sanchez et al., 2011), publicado en la revista “*IEEE Transactions on Computers*” (JCR-ISI, IF 1.608, Q2).
4. A framework for developing home automation systems: From requirements to code (Sánchez et al., 2011), publicado en la revista “*Journal of Systems and Software*” (JCR-ISI, IF 1.293, Q2).

UNIVERSIDAD
POLITÉCNICA DE
CARTAGENA
COMISION DE DOCTORADO

Dña. Mª Francisca Rosique Contreras

Vistos los informes favorables de los Directores de Tesis y el VºBº de la Comisión Académica para la presentación de la Tesis Doctoral titulada: "*Desarrollo integral de aplicaciones domóticas: una perspectiva metodológica*" en la modalidad de "compendio de publicaciones" solicitada por Dña. Mª Francisca Rosique Contreras, la Comisión de Doctorado de la Universidad Politécnica de Cartagena, en reunión celebrada el 24 de febrero de 2012, considerando lo dispuesto en el artículo 33 del Reglamento de Estudios Oficiales de Máster y Doctorado de la UPCT, aprobado en Consejo de Gobierno de 13 de abril de 2011,

ACUERDA

Autorizar la presentación de la Tesis Doctoral a Dña. Mª Francisca Rosique Contreras en la modalidad de compendio de publicaciones.

Contra el presente acuerdo, que no agota la vía administrativa, podrá formular recurso de alzada ante el Sr. Rector-Magnífico de la Universidad Politécnica de Cartagena, en el plazo de un mes a partir de la notificación de la presente.

Cartagena, 28 de febrero de 2012

El Presidente de la Comisión de Doctorado

Fdo. Pedro Sánchez Palma

Universidad
Politécnica
de Cartagena

INFORME DIRECTOR DE TESIS MODALIDAD COMPENDIO DE PUBLICACIONES

D. Pedro Sánchez Palma, Profesor Titular del Departamento de Tecnologías de la Información y las Comunicaciones de la Universidad Politécnica de Cartagena y **D. Manuel Jiménez Buendía**, Profesor Titular del Departamento de Tecnología Electrónica de la Universidad Politécnica de Cartagena,

INFORMAN:

Que los artículos incluidos en la tesis doctoral titulada: "DESARROLLO INTEGRAL DE APLICACIONES DOMÓTICAS: UNA PERSPECTIVA METODOLÓGICA.", presentada por **Dña. María Francisca Rosique Contreras**, bajo la dirección de los Doctores D. Pedro Sánchez Palma y D. Manuel Jiménez Buendía, engloban satisfactoriamente el trabajo de investigación relacionado con la mencionada tesis.

El artículo "**Habitation: A Domain Language for Home Automation**" [1], ha sido publicado en la revista "*IEEE Software*" (JCR-ISI, IF 2.039, Q1). La principal aportación de este artículo es un DSL (Domain Specific Language – Lenguaje Específico de Dominio) específico para el dominio domótico que permite modelar una aplicación domótica de una forma fácil, intuitiva y totalmente independiente de la plataforma.

El segundo artículo, "**A tool for facilitating the teaching of smart home application**" [2], ha sido publicado en la revista "*Computer Applications in Engineering Education*" (JCR-ISI, IF 0.321, Q3). La principal aportación de este artículo es una herramienta para la docencia de los sistemas domóticos basada en el DSL presentado en [2], permitiendo de este modo validar la funcionalidad del DSL en una aplicación real.

El tercer artículo, "**Introducing Safety Requirements Traceability Support in Model-Driven Development o Robotic Applications**" [3], ha sido publicado en la revista "*IEEE Transactions on Computers*" (JCR-ISI, IF 1.608, Q2). Este artículo no trata el dominio domótico en sí, sino que la aportación a la tesis es el enfoque de trazabilidad y la herramienta para la gestión de los modelos de trazabilidad que en él se presenta.

El cuarto artículo, “**A framework for developing home automation systems: From requirements to code**” [4], ha sido publicado en la revista “*Journal of Systems and Software*” (JCR-ISI, IF 1.293, Q2). Este artículo es el más completo y el que mayor aportación realiza a la tesis. En él se presenta un framework que da soporte completo al desarrollo de aplicaciones domóticas, empezando desde la captura de requisitos hasta la generación de código. También hay que indicar que en este artículo se integran conceptos, enfoques y herramientas presentadas en los artículos anteriores, siendo el caso por ejemplo del DSL o el enfoque de trazabilidad.

- [1] M. Jimenez, F. Rosique, P. Sanchez, B. Alvarez, y A. Iborra, «Habitation: A Domain-Specific Language for Home Automation», *IEEE Software*, vol. 26, n°. 4, pp. 30-38, ago. 2009.
- [2] M. Jiménez, P. Sánchez, F. Rosique, B. Álvarez, y A. Iborra, «A tool for facilitating the teaching of smart home applications», *Computer Applications in Engineering Education*.
- [3] P. Sanchez, D. Alonso, F. Rosique, B. Alvarez, y J. A. Pastor, «Introducing Safety Requirements Traceability Support in Model-Driven Development of Robotic Applications», *IEEE Transactions on Computers*, vol. 60, n°. 8, pp. 1059-1071, ago. 2011.
- [4] P. Sánchez, M. Jiménez, F. Rosique, B. Álvarez, y A. Iborra, «A framework for developing home automation systems: From requirements to code», *Journal of Systems and Software*, vol. 84, n°. 6, pp. 1008-1021, jun. 2011.

Los artículos [1], [2], [3], [4] no serán presentados como parte de otra tesis doctoral en la modalidad de compendio de publicaciones. La contribución y trabajo de la doctoranda en la elaboración de todos los artículos citados ha sido relevante.

Por tanto, atendiendo a la calidad y contenido de estas publicaciones, se considera oportuno presentar esta tesis doctoral en la modalidad de “compendio de publicaciones”.

Cartagena, 13 de FEBRERO de 2012

EL DIRECTOR DE TESIS

Fdo.: PEDRO SANCHEZ

EL CODIRECTOR

Fdo.: Nanuel Jimenez
(añadir la firma de un tercer
Codirector, en su caso)

VºBº DEL COORDINADOR

Fdo.: ...
FACION POSGR

**CONFORMIDAD DE DEPOSITO DE TESIS DOCTORAL
POR LA COMISIÓN ACADÉMICA DEL PROGRAMA**

D. Francisco Ortiz Zaragoza, Presidente de la Comisión Académica del Programa
Tecnologías de la Información y Comunicaciones.

INFORMA:

Que la Tesis Doctoral titulada, "DESARROLLO INTEGRAL DE APLICACIONES
DOMÓTICAS: UNA PERSPECTIVA METODOLÓGICA", ha sido realizada por D^a.
María Francisca Rosique Contreras, bajo la dirección y supervisión del Dr. D. Pedro
Sánchez Palma y el Dr. D. Manuel Jiménez Buendía, dando su conformidad a la misma la
Comisión Académica, con la finalidad de que sea presentada ante la Comisión de
Doctorado.

La Rama de conocimiento por la que esta tesis ha sido desarrollada es:

- ☐ Ciencias
- ☐ Ciencias Sociales y Jurídicas
- ☒ Ingeniería y Arquitectura

En Cartagena, a 13 de FEBRERO de 2012

EL PRESIDENTE DE LA COMISIÓN ACADÉMICA DEL PROGRAMA

COMISIÓN DE DOCTORADO

A mis padres.

Agradecimientos

Siempre es difícil recordar a todas las personas que han contribuido durante el largo camino que hay que recorrer para realizar una Tesis Doctoral. En primer lugar, quiero agradecer a mis directores Dr. D. Pedro Sánchez Palma y Dr. D. Manuel Jiménez Buendía su dedicación y apoyo durante este largo camino, especialmente a Pedro Sánchez que me ha guiado y aconsejado de una manera excelente desde que dirigió mi primer PFC.

También quiero agradecer al Dr. D. Andrés Iborra García el haberme dado la oportunidad de integrarme y trabajar en el grupo de Investigación División de Sistemas e Ingeniería Electrónica (DSIE) y a todos mis compañeros del Área de Lenguajes y sistemas, especialmente al Dr. D. Diego Alonso Cáceres, por la orientación, ayuda, compañía y amistad a lo largo de estos años.

Por otro lado, también quiero darle las gracias al Dr. D. Jean-Marc Jézéquel, director del grupo de investigación Triskell en el Institut de Recherche en Informatique et Systèmes Aléatoires en Rennes (Francia), y a todo su grupo, especialmente al Dr. D. Gregory Nain, Dr. D. Sagar Sen, Dr. D. Benoit Baudry, Dr. D. Freddy Muñoz, y D. Juan Cadavid, por haberme acogido cordialmente en Triskell.

A mis padres, Paqui y Vicente, que me han brindado la oportunidad de tener unos estudios y un futuro mejor a costa de muchos sacrificios. Por apoyarme, aconsejarme y aguantar todos los ensayos de mis presentaciones, por ser los directores de la carrera más importante, mi vida.

A mis hermanos Vicente y Jorge y a toda mi familia, por el apoyo incondicional.

A Rafa por darme todo su cariño y apoyo y a todos mis amigos con los que he compartido buenos momentos durante estos estupendos años de realización de la tesis doctoral.

ÍNDICE

1. Resumen / Abstract	1
2. Introducción y objetivos	3
2.1 Objetivos.....	7
3. Estado del arte	9
3.1 Domótica.....	9
3.2 Metodologías de desarrollo de Software.....	13
3.3 Ingeniería Dirigida por Modelos (MDE)	15
3.4 Transformaciones de modelos	19
3.5 Requisitos.....	21
3.6 Trazabilidad	23
3.7 Herramientas	24
4. METODOLOGÍA PROPUESTA.....	27
4.1 DSL HABITATION	28
4.2 Gestión Requisitos	29
4.3 Componentes	32
4.4 Plataforma de Implementación	36
4.5 Trazabilidad	38
5. RESUMENES COMPENDIO DE ARTICULOS.....	43
Habitation: A Domain-Specific Language for Home Automation.....	44
A Tool for Facilitating the Teaching of Smart Home Applications	51
Introducing Safety Requirements Traceability Support in Model-Driven Development of Robotic Applications.....	57
A framework for developing home automation systems: from requirements to code	63
Conclusiones / Conclusions.....	73
Bibliografía.....	77
ANEXO A: CARTAS DE ACEPTACIÓN	81
ANEXO B: ARTÍCULOS.....	85

Índice de Figuras

Figura 1: Visión general de la metodología propuesta.....	6
Figura 2: Aplicaciones de la domótica.....	11
Figura 3: Sistemas domóticos en función del tamaño de la instalación.....	12
Figura 4: Modelo de Ciclo de Vida genérico para MDA.....	18
Figura 5: Esquema básico del funcionamiento de las transformaciones.....	20
Figura 6: Esquema de la metodología propuesta.....	28
Figura 7: Fragmento del catálogo DSL de unidades funcionales.....	29
Figura 8: Fragmento de modelo de aplicación DSL.....	29
Figura 9: Metamodelo de requisitos.....	30
Figura 10: metamodelo de requisitos y modelo de requisitos.....	30
Figura 11: Catálogo de requisitos genérico con un pantallazo de los requisitos en la herramienta Eclipse.....	31
Figura 12: Esquema general del enfoque propuesto en la fase de requisitos.....	32
Figura 13: Metamodelo de componentes en formato TreeEditor.....	33
Figura 14: Metamodelo de componentes en formato EDiagrama.....	33
Figura 15: Esquema del proceso de refinamiento de componentes.....	35
Figura 16: Metamodelo para la plataforma KNX.....	36
Figura 17: Esquema del proceso de generación de código.....	37
Figura 18: Metamodelo de trazabilidad propuesto.....	38
Figura 19: Ejemplos de trazas que pueden representarse con el metamodelo de trazabilidad propuesto.....	39
Figura 20: Informe de trazabilidad <i>Traceability Model #1</i>	41
Figura 21: Informe de trazabilidad <i>Traceability Model #2</i>	41

1. Resumen / Abstract

Los rápidos avances en electrónica, informática y tecnologías de la comunicación (Solé, 2003.) (que conduce a la miniaturización y mejora del rendimiento de los ordenadores, sensores y redes) han dado lugar al desarrollo de nuevas tecnologías en el campo de la domótica (Espinoza, 2011). Las aplicaciones domóticas integran funciones de confort, ahorro energético, seguridad y comunicaciones. El objetivo principal de estos sistemas 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. Por ejemplo KNX (ISO/IEC14543-3-X), Lonworks (ISO/IEC 14908) y X10. Como se indica en (Aenor, 2009), es improbable que se establezca una única tecnología dominante en el campo de la domótica a corto plazo. Además, cada uno de estos estándares proporciona su propio software con el que crear las aplicaciones domóticas y programar los dispositivos. Por lo tanto se debe seleccionar una tecnología en particular (plataforma) en la etapa de diseño inicial, puesto que las herramientas y dispositivos a usar 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.

Por ello, y continuando con la línea de investigación iniciada del Dr. D. Manuel Jiménez en el campo de la domótica (Jiménez, 2009), donde se definió un marco general y los elementos iniciales de un DSL para domótica, se propone aplicar nuevas técnicas de la Ingeniería del Software que permitan la gestión integral del desarrollo del software en todas sus etapas. En concreto para este trabajo de Tesis se propone una metodología que sigue un enfoque de desarrollo dirigido por modelos (MDE) (Bézivin, 2005) (Favre, 2004) junto con un framework de soporte que proporciona los metamodelos y herramientas necesarias en cada nivel.

A continuación, en el capítulo 2 se describen los objetivos estimados para el trabajo de Tesis. En el capítulo 3 se presenta el estado del arte, sobre el que se asienta el desarrollo de la nueva metodología propuesta, que se describe en el capítulo 4, haciendo especial hincapié en la gestión de requisitos y el soporte a la trazabilidad. A continuación, en el capítulo 5 se presentan los resúmenes del compendio de artículos incluidos en esta Tesis. Por último, el capítulo 6 resume las aportaciones realizadas por esta Tesis Doctoral y los resultados obtenidos.

Abstract

Rapid advances in electronics, information and communications technology (Solé, 2003) (leading to miniaturization and improvement of performance of computers, sensor and networking) have given rise to the development of several Home Automation (HA) technologies (Espinoza, 2011). HA applications integrate comfort, energy saving, security and communications functions. The aim of an HA system is to provide homes with a certain degree of intelligence and to improve the quality of life of its inhabitants. Tasks like automatically switching lights and heating, cutting off the supply when gas or water leaks are detected or controlling the home devices remotely from a mobile or a computer through an Internet connection are typical applications of HA domain.

One of the main problems of HA development lies in the fact that there is no agreement in the standard to implement the applications. HA applications and devices currently belonging to different manufactures are isolated from each other thereby creating the main obstacle to HA market growth. Leading companies in this market have adopted several standards and protocols [8]. Some worth mentioning examples are the KNX (ISO/IEC 14543-3-X), Lonworks (ISO/IEC 14908) and X10 technologies. Furthermore, as stated in (Aenor, 2009) it is improbable that there will be a single dominant technology for HA in the short term. Each of these technologies provides its own software suite to create HA applications and program the devices. Hence the particular technology (specific platform) must be selected at the initial design stages, inasmuch as the tools and devices to be used depend on this choice. These facts make the development of HA applications strongly platform dependent, making it very difficult to raise the abstraction level and work with HA domain concepts rather than technology elements.

Therefore, and continuing the research initiated by Dr. D. Manuel Jimenez in the domain of home automation (Jimenez, 2009), which defined a general framework and initial elements of a DSL for home automation, intends to apply new techniques of software engineering to enable the integrated management of software development in all its stages. Specifically, for this thesis, proposes the use of the approach of model-driven development (MDE) (Bézivin, 2005) (Fabre, 2004) together with a set of management tools models ranging from requirements management, traceability, validation and verification, all integrated in a same methodology.

This thesis is structured as follows: Section 2 deals with introducing the objectives. Section 3 presents the state of the art on which rests the development of the proposed new methodology which is described in Section 4, with particular emphasis on requirements management and traceability support. Later, Section 5 presents the abstracts of the articles included in the compendium. Finally Section 6 summarizes the results and contributions of this thesis.

2. Introducción y objetivos

Actualmente existe una amplia variedad de sistemas domóticos (propietarios o abiertos), entre los que destacan, por su alta cuota de mercado y alto crecimiento, las tecnologías Konnex (estandarizada en las normas ISO/IEC14543-3-X y EN50090) y Lonworks (estándares ISO/IEC14908, EN14908 y EIA-709-1).

El problema surge dado el desarrollo de aplicaciones domóticas con estas tecnologías se realiza empleando mayoritariamente las herramientas software proporcionadas por los fabricantes de dispositivos (en caso de sistemas propietarios) o por las asociaciones que dan soporte a la tecnología (en casos de sistemas normalizados). Estas herramientas suelen ser dependientes de la plataforma y orientadas a la generación de código, con una sintaxis poco intuitiva y con un bajo nivel de abstracción por lo que se hace necesario un alto grado de especialización por parte de los desarrolladores y se les obliga a trabajar en un espacio muy cercano a la solución.

Aunque la filosofía de funcionamiento de los estándares más empleados es similar, no se conoce la existencia de herramientas con el alto nivel de abstracción para modelar el sistema domótico a alto nivel y obtener una implementación en uno u otro sistema de manera automática. Esto implica que para el desarrollo de aplicaciones domóticas se utilicen herramientas propias de cada sistema sin posibilidad de portabilidad entre ellas.

Por otro lado, 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 norma general, una solución eficiente y totalmente a medida, antes que diseñar soluciones 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 sistemas desarrollados previamente (incluso simplemente implementados sobre plataformas diferentes).

La metodología tradicional de desarrollo de un sistema domótico sigue los siguientes pasos:

- 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.

- 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 que se desean 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, 2005):

- Interoperabilidad: la variedad de estándares existentes y la falta de un estándar 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 a un nivel muy bajo de abstracción, dependientes de una plataforma específica, lo que en algunos casos complica la modificación o ampliación del sistema, viéndose agravado conforme crecen estos, convirtiéndose en un problema de escalabilidad.
- Robustez: la baja abstracción con la que se desarrollan estos sistemas condicionan la alta probabilidad de fallos que pueden surgir en el software, dado que es un especialista el que se encarga del desarrollo a unos niveles muy próximos al dominio del problema. Esta baja abstracción impide la realización de cambios sin que afecte al resto del sistema, sobre todo ante un cambio inesperado y más aún ante incrementos en complejidad.
- Reutilización: Los sistemas domóticos cuentan con una serie de dispositivos y funcionalidades que se repiten en todas las instalaciones. Sin embargo, con el método de diseño actual es necesario realizar toda la implementación de principio a fin para cada nueva aplicación. 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: La falta de las características anteriores repercuten en una baja productividad y un nivel de calidad que es muy dependiente de la experiencia y formación del ingeniero de 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.

Además, la metodología tradicional de desarrollo cuenta con carencias tales como:

- Ausencia de una notación (gráfica o textual) única para representar los mismos conceptos funcionales que aparecen en las diferentes plataformas.
- Falta de gestión de requisitos de la aplicación.
- Inexistencia de trazabilidad entre los distintos artefactos software generados a lo largo de todo el proceso.
- Falta de herramientas que den soporte al proceso de desarrollo en las distintas tecnologías existentes.

En resumen, la metodología actual presenta numerosos inconvenientes y no satisface los principios básicos de la Ingeniería del Software. Por ello, la utilización de nuevas técnicas de la Ingeniería del Software se plantea como una solución a los problemas asociados al proceso tradicional de desarrollo de aplicaciones domóticas.

Para solventar estas carencias, en esta Tesis se ha desarrollado una nueva metodología que utiliza nuevas técnicas de la Ingeniería del Software, en concreto el enfoque MDE. Gracias a este nuevo enfoque, se puede abordar la creación de herramientas para el control y gestión sistemas domóticos desde una perspectiva mucho más eficaz, obteniendo herramientas más interoperables y fáciles de mantener mediante técnicas que incrementen el nivel de abstracción y la calidad final.

La metodología propuesta se plantea inicialmente como se muestra en la Figura 1. Como se puede observar, los principales pasos que se plantean para obtener un sistema domótico concuerdan con los diferentes niveles MDE y se corresponden con: requisitos domóticos, conceptos específicos del dominio recogidos en el DSL, nivel basado en componentes y, por último, código ejecutable para la plataforma específica. Cada modelo se ha de crear u obtener de acuerdo con su metamodelo, siguiendo el paradigma MDE.

El desarrollo de una aplicación domótica comienza con la captura de los requisitos específicos de esa aplicación. Se realizarán correspondencias (mediante transformaciones) entre los requisitos y el nivel del DSL obteniendo un modelo DSL de la aplicación. Para facilitar esta tarea se cuenta con un catálogo de requisitos reutilizables, asociados a cada uno de ellos su correspondiente fragmento de modelo DSL. De esta manera, se catalogan las soluciones parciales para cada requisito domótico. La idea es que cuando se construya una nueva aplicación, el usuario pueda consultar este catálogo e identificar qué requisitos va a considerar. De esta manera, se permite la reutilización de soluciones adoptadas con el DSL en el desarrollo de aplicaciones anteriores. Esto se ve facilitado por el hecho de que los requisitos domóticos están bien estructurados y es relativamente sencillo asociar a cada requisito

un fragmento de DSL que de manera genérica capture la funcionalidad deseada. Desde el nivel del DSL hasta la generación de código, las transformaciones se realizarán de forma automática y transparente. Cada transformación se refleja en el correspondiente modelo de trazabilidad que posteriormente será procesado por una herramienta que proporciona un conjunto de informes de interés.

Figura 1: Visión general de la metodología propuesta.

2.1 Objetivos

El objetivo de esta Tesis es por tanto doble: por un lado, se quiere demostrar cómo un enfoque de desarrollo dirigido por modelos (MDE) puede ser aplicado de una manera realista y sinérgica en el mundo industrial y, por otro, la creación de una nueva metodología de desarrollo integral de aplicaciones domóticas como solución de los problemas actuales en el desarrollo de sistemas domóticos planteados previamente.

Como objetivo específico de la nueva metodología se plantea la definición de las fases a realizar, los artefactos a obtener, las herramientas y las técnicas a emplear durante todo el desarrollo de un sistema domótico. En este sentido se puede desglosar este objetivo en los siguientes sub-objetivos:

1. Comenzar el proceso de desarrollo con la captura y definición de los requisitos funcionales del nuevo sistema domótico a implementar y proporcionar el soporte necesario para garantizar el cumplimiento de estos requisitos a lo largo de todo el proceso.
2. Integrar el DSL HABITATION como notación gráfica en las primeras fases de la metodología, en particular en:
 - a. Modelado de requisitos domóticos.
 - b. Modelado del sistema a partir del catálogo de requisitos.
3. Incrementar la reutilización de desarrollos anteriores (catálogo de requisitos, catálogo de unidades funcionales).
4. Generar el código de forma automática a partir de modelos gráficos.
5. Gestionar la trazabilidad de los artefactos software involucrados a lo largo de todo el proceso.

En definitiva, esta Tesis propone la creación de una metodología, un entorno asociado y un conjunto de herramientas, cuyo objetivo es dar soporte completo al ciclo de vida del desarrollo de software para sistemas domóticos siguiendo un enfoque MDE.

3. Estado del arte

Hoy día se reconoce ampliamente la necesidad de un enfoque disciplinado en el desarrollo de software y existe un cierto consenso en cuanto a que la Ingeniería del Software ha ganado madurez. Sin embargo, como afirma Sommerville (Sommerville, 2005), la explosión tecnológica que se viene produciendo desde la segunda mitad de los años noventa, y que por ejemplo se puede observar en el auge de múltiples tecnologías de componentes distribuidos y en la evolución de las comunicaciones, no ha sido seguida por una evolución de la misma magnitud en la práctica de la Ingeniería del Software. Todavía se observa en la práctica y se refleja en la literatura de Ingeniería del Software la necesidad de incrementar la calidad del producto software y la productividad en el proceso de desarrollo del mismo.

Diferentes técnicas han permitido a los desarrolladores tratar la creciente complejidad de los sistemas software. En primer lugar metodologías como RUP (Rational Unified Process) (Kruchten, 2004), Metrica 3 (Metrica, 2001), o Extreme Programming (Sillitti, 2010), definen claramente cada paso del proceso de desarrollo. En segundo lugar, mecanismos para elevar el nivel de abstracción, como la programación funcional, orientación a objetos o a aspectos, han permitido a los desarrolladores encapsular la complejidad de los sistemas más fácilmente, y en consecuencia, producir programas más modulares, reusables y extensibles. En tercer lugar, la verificación de software y las pruebas, han ayudado a reforzar la calidad de los sistemas finales.

Sin embargo, no han cubierto todas las necesidades de los desarrolladores para aliviar problemas de cambios, modificaciones durante la fase de desarrollo y mucho menos se adaptan a las necesidades de desarrollos en dominios específicos como es el caso del dominio domótico.

En este sentido el enfoque MDE intenta organizar los nuevos esfuerzos en estas direcciones, proponiendo un marco (1) para definir metodologías, (2) para desarrollar sistemas a cualquier nivel de abstracción y (3) para automatizar y organizar las actividades de prueba y validación. Permitiendo así mejorar el desarrollo de aplicaciones domóticas.

A continuación se presentan brevemente las tecnologías, conceptos, técnicas y herramientas cuyo estudio en profundidad han permitido el desarrollo de este trabajo de Tesis.

3.1 Domótica

En los últimos años, las tecnologías de la información se están integrando en el hogar y la vida cotidiana a gran velocidad. Este proceso ha dado lugar a un nuevo tipo de sistemas reactivos: los sistemas domóticos (Sierra, 2009).

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.

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.*”. 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 cuales una de las más completas es la que se recoge en (Anón 2011), 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 domóticas según el sector de aplicación:

- Domótica, para el sector doméstico (aunque hoy día se ha generalizado para los sectores doméstico y de 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.

3.1.1 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 Figura 2) (Kemp, 2011):

- **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.

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.

Figura 2: Aplicaciones de la domótica.

3.1.2 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 Figura 3).

En viviendas construidas existen tan sólo dos alternativas, el empleo de sistemas basados en transmisión por corrientes portadoras o bien el empleo de 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 derivan de su uso

(poca fiabilidad, alcance limitado, ...). 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.

Figura 3: Sistemas domóticos en función del tamaño de la instalación.

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 distribuidos, especialmente los normalizados **Konnex** y **Lonworks**, cuyo abaratamiento y gran oferta de productos en los últimos años los está haciendo muy populares.

En el caso de los **edificios** las necesidades suelen ser mucho más complejas que en una vivienda. En este ámbito los sistemas en 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.

El sistema de tipo bus más instalado es 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 los sistemas domóticos más relevantes en la actualidad son, en el mercado americano, Lonworks, CEBus y X-10, y en el europeo KNX/EIB. 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.

3.2 Metodologías de desarrollo de Software

El desarrollo de software no es una tarea fácil. Prueba de ello es que existen numerosas propuestas metodológicas que inciden en distintas dimensiones del proceso de desarrollo. Por una parte tenemos aquellas propuestas más tradicionales que se centran especialmente en el control del proceso, estableciendo rigurosamente las actividades involucradas, los artefactos que se deben producir, y las herramientas y notaciones que se usarán. Estas propuestas han demostrado ser efectivas y necesarias en un gran número de proyectos, pero también han presentado problemas en muchos otros. Una posible mejora es incluir en los procesos de desarrollo más actividades, más artefactos y más restricciones, basándose en los puntos débiles detectados. Sin embargo, el resultado final sería un proceso de desarrollo más complejo que puede incluso limitar la propia habilidad del equipo para llevar a cabo el proyecto. Otra aproximación es centrarse en otras dimensiones, como por ejemplo el factor humano o el producto software. Esta es la filosofía de las metodologías ágiles, las cuales dan mayor valor al individuo, a la colaboración con el cliente y al desarrollo incremental del software con iteraciones muy cortas. Este enfoque está mostrando su efectividad en proyectos con requisitos muy cambiantes y cuando se exige reducir drásticamente los tiempos de desarrollo pero manteniendo una alta calidad. Las metodologías ágiles están revolucionando la manera de producir software, y a la vez generando un amplio debate entre sus seguidores y quienes por escepticismo o convencimiento no las ven como alternativa para las metodologías tradicionales.

Un objetivo perseguido durante décadas ha sido encontrar procesos y metodologías, que sean sistemáticas, predecibles y repetibles, a fin de mejorar la productividad en el desarrollo y la calidad del producto software.

La evolución de la disciplina de ingeniería del software ha traído consigo propuestas diferentes para mejorar los resultados del proceso de construcción. Las metodologías tradicionales haciendo énfasis en la planificación, las metodologías ágiles haciendo énfasis en la adaptabilidad del proceso, y por otra parte, la creciente aparición de diferentes tecnologías y plataformas asociadas al desarrollo de sistemas de información, hace demasiado específico el modelado de un sistema. En este sentido, OMG ha propuesto la *Arquitectura Dirigida por Modelos* (MDA) (Mellor, 2002)(MDA, 2012), que garantiza la especificación completa de un sistema en base a modelos, independientes y específicos de una tecnología y plataforma.

3.2.1 Definición de Metodología

En su libro *Object-Oriented Modeling and Design*, Rumbaugh (Rumbaugh et al., 2005) define una metodología de Ingeniería de Software como “un proceso organizado para la producción de software, utilizando una colección de técnicas predefinidas y

convenciones de notaciones. Una metodología usualmente se presenta como una serie de pasos, con técnicas y notaciones asociadas con cada paso”.

Las metodologías se basan en una combinación de los modelos de proceso genéricos (cascada, incremental...). Definen artefactos, roles y actividades, junto con prácticas y técnicas recomendadas.

La metodología para el desarrollo de software en un modo sistemático de realizar, gestionar y administrar un proyecto para llevarlo a cabo con altas posibilidades de éxito. Una metodología para el desarrollo de software comprende los procesos a seguir sistemáticamente para idear, implementar y mantener un producto software desde que surge la necesidad del producto hasta que cumplimos el objetivo por el cual fue creado.

Una definición estándar de metodología puede ser el conjunto de métodos que se utilizan en una determinada actividad con el fin de formalizarla y optimizarla. Determina los pasos a seguir y cómo realizarlos para finalizar una tarea.

Si esto se aplica a la ingeniería del software, podemos destacar que una metodología:

- Optimiza el proceso y el producto software.
- Métodos que guían en la planificación y en el desarrollo del software.
- Define qué hacer, cómo y cuándo durante todo el desarrollo y mantenimiento de un proyecto.

Una metodología define una estrategia global para enfrentarse con el proyecto. Entre los elementos que forman parte de una metodología se pueden destacar:

- Fases: tareas a realizar en cada fase.
- Productos: entradas y salidas de cada fase, documentos generados.
- Procedimientos y herramientas: apoyo a la realización de cada tarea.
- Criterios de evaluación: del proceso y del producto. Saber si se han logrado los objetivos.

El marco de trabajo o *framework* da soporte a la metodología, permitiendo estructurar, planificar y controlar el proceso de desarrollo de un sistema de información. Una gran variedad de estos marcos de trabajo han evolucionado durante los años, cada uno con sus propias fortalezas y debilidades. Una metodología de desarrollo de sistemas no tiene que ser necesariamente adecuada para usarla en todos los proyectos. Cada una de las metodologías disponibles es más adecuada para tipos específicos de proyectos, basados en consideraciones técnicas, organizacionales, de proyecto y de equipo.

El marco de trabajo de una metodología de desarrollo de software consiste en:

- Una filosofía de desarrollo de software, con el enfoque o enfoques del proceso de desarrollo de software.
- Múltiples herramientas, modelos y métodos para ayudar en el proceso de desarrollo de software.

Estos marcos de trabajo están con frecuencia vinculados a algunos tipos de organizaciones, que se encargan del desarrollo, soporte de uso y promoción de la metodología. La metodología con frecuencia se documenta de alguna manera formal.

3.2.2 Ventajas del uso de una Metodología

Son muchas las ventajas que puede aportar el uso de una metodología. A continuación se van a exponer algunas de ellas, clasificadas desde distintos puntos de vista.

Desde el punto de vista de gestión:

- Facilita la tarea de planificación.
- Facilita la tarea del control y seguimiento de un proyecto.
- Mejora la relación coste/beneficio.
- Optimiza el uso de recursos disponibles.
- Facilita la evaluación de resultados y cumplimiento de los objetivos.
- Facilita la comunicación efectiva entre usuarios y desarrolladores.

Desde el punto de vista de los ingenieros del software:

- Ayuda a la comprensión del problema.
- Optimiza el conjunto y cada una de las fases del proceso de desarrollo.
- Facilita el mantenimiento del producto final.
- Permite la reutilización de partes del producto.

Desde el punto de vista del cliente o usuario:

- Garantiza un determinado nivel de calidad en el producto final.
- Genera confianza en los plazos de tiempo fijados en la definición del proyecto.
- Define el ciclo de vida que más se adecue a las condiciones y características del desarrollo.

3.3 Ingeniería Dirigida por Modelos (MDE)

En MDE (Model Driven Engineering) el desarrollo de software se centra en la especificación de modelos y transformaciones y en la reutilización de éstos. La premisa principal de MDE es que los programas se generarán de forma automática a partir de sus correspondientes modelos (Miller et al. 2010). Su objetivo es superar lo que hacen las herramientas CASE, que sólo pueden generar esqueletos de programas, y que tienen poca flexibilidad debido a que no puede ser cambiadas por el usuario. En MDE incluso la semántica de ejecución de los sistemas tendría que especificarse como modelo.

El desarrollo de software en MDE está basado en el refinamiento de modelos a través de transformaciones, es decir, transformar modelos con un mayor nivel de abstracción en

otros de menor nivel de abstracción, hasta alcanzar de forma sencilla la generación de código. De esta manera que el desarrollo del software se torna iterativo, transformando modelos abstractos en otros más concretos, y al final, generando el código de forma automática.

Un modelo es una representación abstracta de la realidad; es una simplificación que muestra ciertos aspectos de lo que se desea modelar, escondiendo aquellos elementos que no son de interés a los que usaran el modelo. Un modelo siempre es conforme a su metamodelo. En un metamodelo están presentes todos los conceptos del dominio que son importantes para el usuario y las relaciones entre ellos. Un metamodelo, por tanto, restringe completamente los elementos que pueden formar parte de un modelo de un sistema en particular y sus relaciones. De esta forma, un modelo siempre es conforme a su meta-modelo (por definición). Sin embargo, el concepto de metamodelo no es un concepto absoluto, sino que depende del nivel de abstracción utilizado. Por tanto, lo que con un determinado nivel es un metamodelo se puede convertir en modelo en otro.

En general, los modelos son útiles para resaltar los aspectos importantes de la realidad representada, dejando de lado los elementos que puedan distraer la atención de aquello que realmente interesa. De forma específica, han sido aprovechados dentro la ingeniería para representar los problemas planteados y sus posibles soluciones; ya que la abstracción realizada en el modelo se ha utilizado como base para una mejor comunicación entre los diferentes participantes del problema, así como para una mejor comprensión del mismo.

Un proceso MDE debe especificar la secuencia de los modelos que se tienen que desarrollar, y cómo derivar un modelo a partir de otro, por lo general del nivel de abstracción inmediatamente superior. Proporcionando a los desarrolladores una metodología como ésta, podrán saber en cada momento a lo largo del proceso de desarrollo, qué se debe hacer en cada paso y cómo conseguirlo.

El problema es que MDE proporciona una estrategia general que se tiene que seguir durante el desarrollo del software, pero no define técnicas a utilizar o fases del proceso, así como ningún tipo de guía metodológica. Existen algunos enfoques que aplican MDE tales como: MDA (Model Driven Architecture (Mellor et al.,2002.)), Factorías del Software (Demir 2006) o MIC (Model Integrated Computing) (Sztipanovits 2005).

MDA es la propuesta con más fuerza en el ámbito de desarrollo software. Es una iniciativa de la OMG (Object Management Group) y se basa en estándares como XMI (XML Metadata Interchange), MOF (Meta Object Facility) y CWM (Common Warehouse Metamodel). La idea clave de MDA es que si el desarrollo está guiado por modelos software, se obtendrán beneficios de productividad, portabilidad, interoperabilidad, mantenimiento y documentación.

A continuación se presenta un pequeño resumen de MDA, ya que es el enfoque seleccionado en esta Tesis.

3.3.1 MDA

Según la guía de MDA (MDA, 2012), 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.

Para este fin, el marco de trabajo MDA especifica tres niveles de abstracción que nos proporciona tres puntos de vista diferentes, los cuales a su vez generan un modelo que representa los resultados de la aplicación de cada punto de vista. Los tres niveles MDA son:

- Nivel de Modelo Independiente de Computación (CIM – Computation Independent Model). Este punto de vista está centrado en el dominio del sistema así como en los requerimientos, detalles de la estructura y procesamiento del sistema.
- Nivel de Modelo Independiente de Plataforma (PIM – Platform Independent Model). Esta vista es la encargada de mostrar la especificación del sistema tomando en cuenta no solo las especificaciones de funcionamiento propias del sistema – especificadas 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 ó método de implantación escogido para la representación informática.
- Nivel específico de plataforma (PSM). Esta vista combina el punto de vista independiente de plataforma con los detalles y características propias del uso de una plataforma de desarrollo. En el Modelo Especifico de Plataforma (PSM – Platform Specific Model) se puede observar la manera en la cual un sistema usa las herramientas de la plataforma para el cumplimiento de los objetivos trazados en la etapa de especificación inicial.

Entre los diferentes modelos construidos en MDA existe una estrecha relación. Los modelos más abstractos son la base para la construcción de los modelos específicos así como los modelos específicos son los que soportan los modelos de un nivel de abstracción mayor. Esta relación es representada en esta arquitectura por las Transformaciones entre Modelos, las cuales constituyen una de las características fundamentales de este enfoque.

3.3.2 Proceso de Desarrollo con MDA

En este trabajo de Tesis se considera cada fase del ciclo de vida como un nivel conceptual. Dentro de cada fase es posible a su vez diferenciar subniveles conceptuales. Las transformaciones pueden darse tanto entre modelos de un nivel superior a uno inferior, como entre modelos de un mismo nivel.

En la Figura 4 se muestra el Ciclo de Vida genérico para MDA, a continuación se detallan cada una de las etapas:

La primera etapa corresponde a la fase de análisis de requisitos. Su objetivo es obtener una especificación adecuada de cuáles son los requisitos del sistema que se desea

desarrollar. Esta es una de las fases más importantes, ya que disponer de una gestión de requisitos completa y precisa evitará propagar errores a las siguientes fases. Esta etapa se suele encajar en el nivel CIM de la arquitectura MDA.

Una vez obtenidos los requisitos del sistema, se pasa a la fase de nivel PIM, en la que se construirán los modelos que describen la funcionalidad del sistema de forma independiente a cualquier plataforma.

En la siguiente etapa, los modelos PIM se transformarán en modelos PSM. Estos modelos serán usados como entrada para la fase de generación de código.

A continuación sigue la fase de pruebas del sistema, si los resultados son erróneos puede implicar el regreso a la fase anterior de generación de código o a fases superiores (CIM-PIM-PSM).

Es aconsejable que durante todo el proceso se realice una gestión de trazabilidad de los artefactos involucrados en el proceso.

Hay que tener en cuenta que MDA no es por si mismo un método que define técnicas, etapas, artefactos, etc., solamente proporciona la infraestructura tecnológica y conceptual con la que construir estos métodos.

En definitiva, es necesario que se construyan métodos precisos que proporcionen las pautas a seguir y utilizar por los desarrolladores. Además, para el desarrollo de una aplicación dentro del marco de trabajo MDA son necesarios también contar con los siguientes elementos: (1) un conjunto de modelos con sus correspondientes metamodelos, (2) un lenguaje y herramientas de modelado, (3) un formato de almacenamiento e intercambio de modelos, (4) un lenguaje y herramientas para transformaciones.

3.3.3 Metamodelado y DSL

El objetivo principal de MDE es construir software a partir de modelos, desplazando el uso tradicional del código fuente como protagonista principal de los procesos de desarrollo. La idea compartida por todos los paradigmas englobados dentro de MDE es la conveniencia de que se empleen lenguajes de más alto nivel de abstracción que los lenguajes tradicionales de programación. Estos lenguajes mantienen un concepto más cercano al dominio de la aplicación. Estos lenguajes pueden ser lenguajes genéricos o

Figura 4: Modelo de Ciclo de Vida genérico para MDA

bien específicos de un dominio, estos últimos son los llamados Lenguajes Específicos del Dominio (DSL) (Mernik et al. 2005),(Tolvanen 2011).

La teoría subyacente a la creación DSL se denomina *metamodelado* y contiene diferentes aspectos que abarca: *metamodelo*, *sintaxis abstracta*, *sintaxis concreta*, *semántica* y *transformaciones* y sus respectivas relaciones (Mernik et al. 2005).

Un metamodelo define la sintaxis abstracta de un lenguaje que establece los conceptos y relaciones entre ellos, e incluye las reglas que determinan qué es un modelo bien formado. El metamodelo deber ir acompañado de una definición de la sintaxis concreta o notación para expresar los modelos que conforman a la sintaxis abstracta. La semántica se refiere al significado de los conceptos y de las relaciones en el lenguaje, lo cual es necesario para comprender el lenguaje.

En resumen, un metamodelo define los conceptos de un lenguaje y las relaciones que entre ellos se establecen (Jiang and Yun, 2011). Si tomamos como ejemplo un mapa, el metamodelo define el lenguaje que se usa para definir el mapa, es decir, el metamodelo es su leyenda

Al igual que la relación que se establece entre un sistema y su modelo es de representación (el modelo representa al sistema), al introducir los metamodelos se estable una nueva relación de conformidad entre el modelo y el metamodelo: un modelo es conforme a un metamodelo. Esta relación indica que un metamodelo ha sido creado de acuerdo con los conceptos y reglas definidos en el metamodelo.

Por otro lado entre metamodelo y lenguaje se establece una relación de definición. A su vez, un lenguaje de modelado se define como un conjunto de modelos (Clark et al., 2008), en particular el conjunto formado por todos los modelos que ese lenguaje puede formar, los cuales pertenecen a él.

3.4 Transformaciones de modelos

En el contexto MDE una transformación es el proceso de generación de un modelo destino a partir de uno origen de acuerdo con una definición de transformación (Czarnecki and Helsen, 2003). En (OMG, 2012) se define el concepto de transformación como “*la generación automática de un modelo destino a partir de un modelo origen, de acuerdo con una definición de la transformación. Una definición de la transformación es el conjunto de reglas de transformación que juntas describen cómo un modelo en un lenguaje origen puede ser transformado en un modelo en el lenguaje destino. Una regla de transformación es la descripción de cómo una o más construcciones en el lenguaje origen pueden ser transformadas en una o más construcciones en el lenguaje destino*”.

En resumen, una transformación es un proceso descrito por una *definición de transformación*, la cual consiste de un conjunto de *reglas de transformación*, que son ejecutadas por una *herramienta de transformación*. Las transformaciones son una parte integral dentro de MDE. Una herramienta de transformaciones puede realizar transformaciones de modelos de forma automática siempre y cuando cuente con las reglas precisas de cómo debe realizar la transformación (Favre, 2004) (Mens and vanGorp, 2006)(Rozenber, 1997).

Figura 5: Esquema básico del funcionamiento de las transformaciones.

Sin embargo, una transformación no es sólo un conjunto de reglas sino que también debería llevar asociada una serie de características deseables, en el capítulo 7 de la guía MDA (MDA, 2012.) se exponen estas características, de mayor a menor importancia son:

- Configuración: es la principal de las características y describe la posibilidad de poder configurar una transformación antes de ser usada. Es muy importante que una herramienta dé la posibilidad de definir transformaciones configurables que permiten desarrollar transformaciones entre modelos más potentes y versátiles.
- Trazabilidad: es la capacidad de poder seguir rastro a un elemento del modelo destino hasta el elemento o elementos del modelo origen que la generaron.
- Consistencia Incremental: es la capacidad para mantener los cambios manuales realizados en elementos del modelo destino aunque éste vuelva a ser regenerado con la transformación que lo originó.
- Bidireccionalidad: esta característica se define como la capacidad de ejecutar una transformación que se haya definido bidireccionalmente. Una transformación que cumpliera esta característica permitiría que su aplicación al modelo destino diera como resultado el modelo origen.
- Las transformaciones pueden ser tanto horizontales, entre modelos del mismo nivel de abstracción, como transformaciones verticales, entre modelos de distintos niveles de abstracción.

Las transformaciones pueden clasificarse desde varios puntos de vista, dependiendo del lenguaje, la abstracción, etc. Una posible clasificación se puede hacer distinguiendo dos tipos, dependiendo de la fase en que se realiza y el tipo de artefacto generado:

- **Modelo a modelo (Model-to-Model (M2M))**: siguiendo la definición provista, es el proceso de convertir un modelo en otro modelo del mismo sistema, se realiza especificando la transformación de un objeto desde un modelo origen a uno o más objetos en un modelo destino, siguiendo distintos enfoques. Estas transformaciones pueden ser a su vez :

- Transformaciones horizontales, si el modelo origen y destino pertenecen al mismo nivel de abstracción. Estas transformaciones se utilizan para realizar un refinamiento entre modelos, por ejemplo de un CIM a otro CIM o de PIM a PIM.
- Transformaciones verticales, si el modelo origen y destino pertenecen a distintos niveles de abstracción.
- 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.

En la actualidad, no existe un lenguaje estándar que sirva para definir las transformaciones modelo-a-modelo ni modelo-a-texto. Cada compañía desarrolla su propio lenguaje y su propia herramienta.

La OMG ha propuesto QVT (Query View Transformation) (Mens and vanGorp, 2006) como estándar para transformaciones de modelos. Una transformación QVT se define como un conjunto de reglas, las cuales tienen un patrón de entrada y unos elementos MOF de salida. Además, dichas reglas pueden tener: unas condiciones, que determinan si se pueden activar las reglas; y una parte imperativa, que se ejecuta cuando se activan las reglas. Siguiendo parte de la especificación de QVT, el lenguaje más conocido es ATL (Atlas Transformation Language). Hay incluso lenguajes de transformación QVT visuales, como por ejemplo UMLX. Otro enfoque con bastante peso es el de las transformaciones de grafos, donde AGG (Algebraic Graph Transformation) [31] es la herramienta más utilizada.

En transformaciones modelo-a-texto se pueden encontrar: MOFScript que es un lenguaje mitad declarativo mitad imperativo propuesto como estándar para definir este tipo de transformaciones. JET (Java Emitter Templates) (JET, 2012) utiliza una sintaxis basada en etiquetas para describir plantillas que expresan el código que se desea generar.

3.5 Requisitos

La gestión de requisitos es un componente vital en el desarrollo de un proyecto software ya que provee la dirección y el alcance del proyecto que se quiere desarrollar. El uso de herramientas para auxiliar la gestión de requisitos se ha convertido en un aspecto importante de la Ingeniería de Software.

Considerando el tamaño y la complejidad del desarrollo, las herramientas vienen siendo algo esencial. Las herramientas que los gestores de requisitos utilizan para automatizar los procesos han disminuido el trabajo duro en el mantenimiento de requisitos y reduciendo errores del proceso de desarrollo.

La Ingeniería de Requisitos es la disciplina de la Ingeniería de Software donde se identifica el propósito del sistema, dirección y alcance. Consiste en un conjunto de actividades y transformaciones que pretenden comprender las necesidades de un sistema software y convertir la declaración de estas necesidades en una descripción completa, precisa y documentada de los requerimientos del sistema siguiendo un determinado

estándar. Los requisitos constituyen el enlace entre las necesidades reales de los clientes, usuarios y otros participantes vinculados al sistema con el sistema real.

Se han propuesto muchas taxonomías sobre tipos de requisitos (IEEE, 1999) (Loucopoulos and Karakostas, 1995), aunque existe un cierto consenso en que los requisitos se pueden clasificar en:

- Requisitos funcionales: aquellos requisitos relativos a las capacidades y servicios que el sistema o el software debe proporcionar.
- Requisitos no funcionales: aquellos relativos a los atributos de calidad del sistema y/o del software en el desempeño de sus funciones, por ejemplo, rendimiento, disponibilidad, portabilidad o seguridad.
- Restricciones de diseño: son las condiciones existentes para el diseño.

Estos requisitos una vez establecidos y documentados, pueden sufrir cambios continuos, en este sentido, no basta con tratar el análisis de los mismos sino que es más importante centrarse en su gestión, es decir, el seguimiento respecto a los cambios que se generan durante el ciclo de vida del proyecto y las herramientas de gestión de requisitos que ayudan y/o automatizan estas tareas. Es necesario gestionar estos cambios para asegurar que la calidad de los mismos se mantenga, los problemas suscitados por los cambios de requisitos podrían incurrir en altos costes (Sommerville and Sawyer, 1997).

La gestión de requisitos se puede entender como el proceso encargado de la identificación, asignación y seguimiento de los requisitos, incluyendo interfaz, verificación, modificación y control a lo largo de todo el proceso de desarrollo.

Las tareas principales de la gestión de requisitos son la de (1) captura de los requisitos, (2) documentación de los requisitos capturados, (3) verificación de los requisitos y (4) gestión de cambios.

Tradicionalmente, la etapa de captura de requisitos se realiza en formato textual, realizando encuestas e interaccionando con los clientes. Estos requisitos acaban documentándose y el programador se encargaba de tenerlos en cuenta.

Gracias al enfoque MDE se pueden utilizar modelos para capturar los requisitos, quedando integrado todo dentro del mismo proceso. De esta manera la primera fase del desarrollo comenzaría con la captura de los requisitos (incluso en un formato gráfico) y a partir de ellos se obtendrían automáticamente el resto de artefactos involucrados en el proceso.

3.5.1 Reutilización de requisitos

El propósito de la reutilización de requisitos es identificar descripciones de sistemas que puedan ser reutilizadas (en su totalidad o en parte) con un mínimo número de modificaciones, de manera que se reduzca el esfuerzo total de desarrollo (Cybulsky and Reed, 2000). Este nivel de reutilización puede aportar grandes beneficios. Como se ha comentado anteriormente la forma más utilizada de representación de requisitos es el lenguaje natural, pero además de los problemas inherentes al lenguaje natural, en (Laguna et al., 2004) se constata que la diversidad de formatos de requisitos es una

restricción para su reutilización y por otro lado en (Robertson and Robertson, 2006) se afirma que, cualquier diagrama o especificación que permita hacer los requisitos visibles incrementan la posibilidad de reutilización.

Con estas hipótesis de partida se ha considerado más que necesario que la fase de captura de requisitos se realice haciendo uso de modelos gráficos que permitan a su vez catalogar y almacenar los requisitos para su uso y reutilización en futuros desarrollos. Para ello es necesaria una herramienta de soporte para la gestión de requisitos en general.

Varios autores (Monzón and Dueñas, 2004) (Cerón et al., 2005) señalan la ausencia de soluciones prácticas a la reutilización de requisitos con herramientas comerciales y presentan guías para reutilizar requisitos y definir metamodelos que permitan la clasificación de estos.

3.6 Trazabilidad

Como ya se ha indicado, en el enfoque MDE, un artefacto software es visto como un modelo. Tareas típicas como gestión de requisitos, transformaciones de modelos, producción de código, integración de aplicaciones o interoperabilidad, son realizadas directamente con modelos. En general, se producen una serie de pasos donde la información de cada uno de ellos debe ser almacenada para poder ser consultada posteriormente (Olsen and Oldevik, 2007). Es por ello que la trazabilidad es una cuestión clave en estos tipos de procesos y motivo por el cual debe de proveerse la gestión de la trazabilidad y las herramientas que den soporte dentro de la metodología de desarrollo.

Por otra parte, como en cualquier escenario del campo de la Ingeniería del Software en el que existe una manipulación de un artefacto software, la capacidad de describir y consultar las operaciones de manipulación que se han realizado sobre un determinado artefacto puede ser relevante para otras tareas relacionadas (Kolovos et al., 2006).

Para conseguir soporte a trazabilidad en el proceso de desarrollo software han de tenerse en cuenta diversas tareas (Ramesh and Jarke, 2001):

- Definición de la traza: para indicar los tipos de objetos del sistema que pueden ser trazados, y que información se va a definir en una traza.
- Producción de la traza: para indicar qué actividades, acciones, decisiones y eventos ocurridos a lo largo del desarrollo del software generan trazas.
- Extracción de la traza: para indicar cómo las trazas generadas en la producción, pueden ser consultadas con el fin de obtener cierta información, como por ejemplo información para validación de requisitos o mantenimiento software.
- Verificación de la traza: para mantener la integridad del conjunto de objetos y trazas.

Para facilitar una gestión eficiente de la trazabilidad es deseable el uso de herramientas de soporte que realice de forma integrada al resto del desarrollo, todas las tareas mencionadas anteriormente (Behrens, 2007).

3.6.1 Trazabilidad de requisitos

Realizar el seguimiento a los requisitos a lo largo de todo el proceso no es tarea fácil. Todo artefacto software cambia en el tiempo por la evolución de las necesidades del usuario y más aún en un desarrollo dirigido por modelos, donde los modelos sufren continuas transformaciones. Para minimizar el impacto causado por dicha evolución se viene utilizando diferentes técnicas y modelos de trazabilidad que permiten lograr una mayor calidad en los productos software (Almeida et al., 2006).

La trazabilidad de requisitos es clave para conseguir una exitosa gestión de requisitos. Dicha importancia no se ve reflejada en un consenso respecto de las prácticas con el que el proceso de trazabilidad ha de llevarse a cabo (Ramesh and Jarke, 2001). No existen estándares asociados al proceso de trazabilidad que ayuden a determinar qué tipos de artefactos y de enlaces se han de considerar. Esto provoca la paradoja de que a pesar de la importancia de este proceso y de las múltiples herramientas de gestión de requisitos, no se provean soluciones adecuadas para configurar la trazabilidad de acuerdo a las necesidades específicas del proyecto (Cleland-Huang et al., 2009).

En la Ingeniería de Requisitos, la Guía para la Especificación de Requisitos Software de IEEE indica que una especificación de requisito software es trazable si el origen de cada requerimiento es claro y si facilita la referencia de cada requisito en desarrollos futuros o mejoras.

Basado en esta definición, en (Cleland-Huang et al., 2009) se describe Trazabilidad de Requisitos como la capacidad de describir y seguir la vida de un requerimiento en ambos sentidos, hacia sus orígenes o hacia su implementación, a través de todas las especificaciones generadas durante el proceso de desarrollo de software.

En trazabilidad de requisitos podemos identificar dos actividades (Gotel and Finkelstein, 1994): (a) configurar la trazabilidad respecto de las necesidades del proyecto y (b) especificar y explotar la información de trazabilidad durante el desarrollo y mantenimiento del software.

Es muy útil integrar la trazabilidad de requisitos en el mismo entorno de desarrollo utilizado para poder utilizar los artefactos involucrados directamente (Melby, 2007). Al proporcionar un único entorno se evitan posibles problemas de sincronización y de cambio de contexto, así se puede realizar la gestión de requisitos y el modelado de la aplicación en el mismo entorno.

3.7 Herramientas

Eclipse es el entorno de desarrollo elegido para realizar este trabajo de Tesis. Eclipse es un entorno de libre distribución, ampliable y configurable gracias a un diseño modular, que permite la adición de extensiones mediante el uso de plugins.

Aunque está vinculado a Java, existen numerosos plugins que permiten desarrollar software en otros lenguajes de programación. Eclipse es actualmente el único entorno de desarrollo gratuito que proporciona el soporte necesario para realizar un desarrollo basado en modelos y el más extendido en esta comunidad. A continuación se describen brevemente los principales plugins de Eclipse que se han utilizado para desarrollar la presente Tesis:

- **Eclipse Modelling Framework (EMF).** EMF es el plugin principal del soporte MDE de Eclipse, ya que contiene la implementación de parte del estándar MOF (concretamente de la parte básica Essential-MOF (**EMOF**)) del OMG. El proyecto EMF contiene otros plugins que permiten especificar y realizar consultas sobre los elementos de un modelo EMF y su contenido, así como para verificar la integridad de los modelos realizados, i.e. si el modelo es conforme a su metamodelo.
- **Eclipse Graphical Modeling Framework (GMF)** proporciona soporte a la generación e infraestructura en tiempo de ejecución para el desarrollo de editores gráficos basados en *EMF* y *GEF* (Graphical Editing Framework). *GMF* está dividido en dos componentes principales: la infraestructura que da soporte en tiempo de ejecución y las herramientas para generar editores capaces de utilizar ese *runtime*.
- **Java Emitter Template (JET)** es una herramienta de *EMF* que se utiliza típicamente para la generación de código. La generación de código es una parte importante de *MDD*, ya que la meta de *MDD* es describir los sistemas software como modelos abstractos y después definir una transformación de estos modelos a código. Sería deseable que este proceso se realizara dentro de lo posible de manera automática, en pos de un proceso más robusto y una mejor calidad del software. Las transformaciones automáticas como las realizadas mediante las plantillas de *JET* pueden lograr esto.
- **ATLAS Transformation Language (ATL).** ATL es un lenguaje de transformación de modelos mixto (declarativo/imperativo), que está basado en el lenguaje OCL. ATL soporta transformaciones con múltiples modelos de entrada y múltiples modelos de salida. Una transformación entre modelos descrita en ATL consta de un conjunto de «reglas» que especifican cómo se transforman los elementos del metamodelo de origen al metamodelo de destino y de un conjunto de funciones auxiliares. Además, ATL soporta todas las operaciones sobre colecciones definidas en OCL y la navegación por el modelo origen. ATL no es una implementación de QVT, sino que fue la respuesta del grupo de investigación ATLAS del INRIA al «Request for Proposals» realizado por el OMG para definir el estándar QVT.

4. METODOLOGÍA PROPUESTA

Siguiendo con la definición y características de una metodología como se detalló en el capítulo 3, y teniendo en cuenta que la nueva metodología integral de desarrollo de sistemas domóticos que se ha desarrollado en esta Tesis sigue el enfoque MDA, se pueden distinguir tres grandes fases metodológicas a tener en cuenta: (1) fase de análisis y definición de requisitos (Nivel CIM), (2) fase de diseño (Nivel PIM) y (3) fase de implementación (Nivel PSM).

La Figura 6 muestra el marco completo de desarrollo de sistemas domóticos que da soporte a esta metodología. En ella pueden observarse (parte derecha) los distintos niveles MDA correspondientes a (1) requisitos domóticos, (2) lenguaje específico de dominio, (3) modelo integrado de componentes, y (4) generación de código para una plataforma específica. Cada modelo construido es conforme a su metamodelo. Entre el nivel de requisitos y el DSL las correspondencias se establecen de forma manual. De esta manera, quedan catalogadas las soluciones parciales a cada requisito domótico. Al construir una aplicación nueva, el usuario debe inspeccionar dicho catálogo identificando los requisitos que va a reutilizar del mismo. Al hacerlo, reutiliza las soluciones adoptadas con el DSL en el desarrollo de aplicaciones anteriores. Desde el nivel del DSL hasta la generación de código, las transformaciones se realizan de manera automática (por el momento sólo para la plataforma KNX/EIB). Cada transformación de un nivel al inferior, bien sea manual bien automatizada, queda registrada en el correspondiente modelo de trazabilidad (parte izquierda de la Figura 6), también conforme a un metamodelo que se describe a continuación. Los modelos de trazabilidad, son posteriormente procesados por la herramienta RTT (Requirement Tracer Tool) que proporciona al usuario informes de trazabilidad útiles para distintas situaciones.

A continuación se presentan cada una de estas fases y niveles con un mayor grado de detalle, comenzando con la explicación del DSL dado que es en él donde se definen los conceptos del dominio que han influenciado en el resto de la metodología.

Figura 6: Esquema de la metodología propuesta.

4.1 DSL HABITATION

La definición de este DSL se inició en el seno de un proyecto de investigación del grupo DSIE y fue presentado en la Tesis del Dr. D. Manuel Jiménez (Jiménez, 2009) y en el artículo (Jiménez et al., 2009) incluido en el compendio de publicaciones de este trabajo de Tesis. En ellos se definen los conceptos del dominio domótico a utilizar y se identificaron los elementos que debían incluirse en este lenguaje. Este trabajo fue punto de partida para esta Tesis, además ha sido mejorado gráficamente, se han añadido restricciones de integridad y se ha incorporado al *framework*. Uno de los conceptos claves que hace que este DSL sea idóneo para modelar sistemas domóticos es el concepto de “Unidad Funcional”, este concepto ha marcado un punto de inflexión entre los desarrollos tradicionales y esta nueva metodología ya que permite dotar a las primeras fases de la metodología de una total independencia de la plataforma final de implementación.

Una “Unidad Funcional” es la mínima parte funcional en la que se puede descomponer un dispositivo domótico. Como ya se ha mencionado en capítulos anteriores, el principal problema en los desarrollos de sistemas domóticos es la inexistencia de un estándar *de facto*, por lo que cada plataforma, estándar o fabricante proporcionan sus propios dispositivos, aunque todos ellos intentan cubrir las mismas necesidades funcionales. Por ejemplo podemos encontrar un dispositivo que ofrezca cuatro salidas de regulación de luminosidad y otro dispositivo diferente con tan solo dos salidas de regulación de luminosidad. Dependiendo de la plataforma de implementación final se deberán utilizar dos dispositivos o sólo uno. Este tipo de decisiones comprometían la independencia de las primeras fases. Por esta razón se ha decidido trabajar con unidades funcionales en lugar de trabajar directamente con dispositivos, de modo que se pueda asegurar la independencia de plataforma en las primeras fases de desarrollo.

La sintaxis abstracta de este DSL se recoge en un único metamodelo (para más detalle véase (Jimenez et al., 2009)), aunque para obtener una mayor modularidad se ha optado por separar el metamodelo en dos partes, la parte utilizada para el catálogo de unidades funcionales y una segunda parte para la aplicación domótica. De este modo, el catálogo se modela de manera independiente a la aplicación, así se posibilita su reutilización de las unidades funcionales definidas en el catálogo y se asegura poder mantener el catálogo actualizado en todo momento sin necesidad de modificar el metamodelo. En la Figura 7 se puede ver una proyección de un ejemplo de catálogo de unidades funcionales realizada con la interfaz gráfica que da soporte a la sintaxis concreta de la parte del catálogo. La Figura 8 muestra la parte de la aplicación.

Figura 7: Fragmento del catálogo DSL de unidades funcionales.

Figura 8: Fragmento de modelo de aplicación DSL.

4.2 Gestión Requisitos

La captura de requisitos se sitúa en el nivel CIM definido por MDA y se corresponde con la primera fase de la metodología propuesta. Aquí se recogen los requisitos solicitados por el usuario y que el sistema domótico debe satisfacer.

En esta Tesis se ha integrado la gestión de requisitos dentro de la metodología siguiendo el enfoque MDE, lo que ha permitido una reutilización sistemática de los requisitos y su posible trazabilidad. La aplicación de las técnicas MDE han obligado a tener que definir un metamodelo de requisitos y las transformaciones M2M entre los niveles inferiores de la metodología.

Por otro lado, en el desarrollo de sistemas domóticos es habitual que los requisitos básicos estén repetidos o que sean muy parecidos entre distintos proyectos. Por esta razón se ha creado un catálogo que permita reutilizar los requisitos entre los distintos desarrollos de sistemas domóticos, quedando por tanto el metamodelo como se muestra en la Figura 9.

Figura 9: Metamodelo de requisitos.

Este metamodelo tiene una versatilidad suficiente, con una estructura simple que podría ser aplicado también a otros dominios. El elemento raíz es *Catalogue*, donde se incluyen el conjunto de requisitos. Cada uno de estos requisitos incluye un nombre, una descripción y un procedimiento de validación para el usuario con el fin de validar si el sistema cumple la funcionalidad esperada.

Los requisitos normalmente se encuentran fuertemente relacionados entre ellos, por lo que se ha incorporado la EClass *Relationship* que permite relacionar varios requisitos entre sí. El tipo de relación entre requisitos se identifica con el elemento *TypeOfRelationship*.

La Figura 10 muestra un ejemplo del catálogo de requisitos basados en la estructura de este metamodelo en el entorno Eclipse.

Figura 10: metamodelo de requisitos y modelo de requisitos.

Como primera aproximación, se ha propuesto un conjunto básico de requisitos para los sistemas domóticos (la Figura 11 muestra un extracto). Estos requisitos están divididos en cuatro categorías básicas: confort de iluminación (iluminación), el confort

motorizado (para dispositivos movidos por motores, como puertas, persianas, proyectores, etc), el confort de climatización, y la seguridad. Requisitos de comunicación sólo se han incluido dentro de la seguridad para obtener información acerca de los tipos de alarma enviados a móviles por SMS.

REQUIREMENTS CATALOGUE					
CONFORT - LIGHTING					
ID	NAME	DESCRIPTION	VALIDATION PROCEDURE	TARGETS	TYPE OF RELATIONSHIP
Req.1	ON/OFF	Control of Light through a push-button or any other domotic mechanism.	Apply push button PB to verify the correct functioning of the light.	None	—
Req 2	Dimmer	Light intensity regulation through the mechanism, short push for ON/OFF, longer push for Dimmer Up or Down.	Push the mechanism (ej: push-button) short for ON/OFF and longer for regulation in order to verify correct functioning.	None	—
Req.17	Automatic-Day Light Sensor	Automatic Light Switching through a daylight sensor.	Wait for daylight sensor to detect the diminishing light outside, and see if the light gets turned on.	Req1, Req2	Requires at least one of them.
Req.18	Automatic-Timer	Automatic Light Switching through time programmation (clock).	Program the timer and verify if the light turns on at the programmed time.	Req1, Req2	Requires at least one of them.
Req 3	Automatic-Presence	Automatic Light Switching through a presence detector.	Walk near the presence detector to verify the correct functioning of the light.	Req1, Req2	Requires at least one of them.
Req Comfort Illuminat.	Automatic-Presence&Timer	Automatic Light Switching through a presence detector when timer has been programmed to function.	Walk near the presence detector when the time has been programmed for functioning to verify the correct functioning of the light.	Req1, Req2, Req3, Req.18	Requires all of them.

Figura 11: Catálogo de requisitos genérico con un pantallazo de los requisitos en la herramienta Eclipse.

Cada uno de los requisitos tiene asociado un fragmento de modelo DSL donde se representa en notación gráfica el requisito. Esta asociación se ha realizado de forma manual y la selección actual de los requisitos del catálogo también se realiza de forma manual, aunque se está desarrollando (en colaboración con la Universidad de Murcia) un plugin en java que permitirá en un futuro poder realizar la selección de una forma más fácil. De esta manera, a la hora de desarrollar una aplicación sólo se debe seleccionar los requisitos necesarios del catalogo y automáticamente se genera un modelo inicial de aplicación en notación DSL (véase Figura 12). Este modelo puede ser modificado posteriormente para refinar el modelado de la aplicación. Cada una de las correspondencias entre requisito y elementos del DSL se han trazado con el fin de poder realizar un seguimiento de estos requisitos a lo largo del ciclo de vida de desarrollo del software para el sistema domótico y poder gestionar un control de cambios exhaustivo.

Figura 12: Esquema general del enfoque propuesto en la fase de requisitos.

4.3 Componentes

Para el nivel PIM del *framework*, se ha decidido utilizar un modelo de componentes que sirva de nivel intermedio para el desarrollo y que haga de punto de unión con otros sistemas reactivos (por ejemplo el robótico, redes de sensores, visión artificial, ...)

En esta fase se deben tomar decisiones muy importantes en cuanto a metodología, el principal problema que determina la estrategia a seguir en esta fase viene marcado por la fuerte dependencia existente en cada una de las plataformas.

Como se ha dicho en numerosas ocasiones, para asegurar la independencia de la plataforma en las primeras fases de desarrollo, se optó por utilizar unidades funcionales en lugar de dispositivos. El problema de trabajar con unidades funcionales surge en el salto de un modelo totalmente independiente de plataforma a un modelo de plataforma donde se modelan dispositivos reales.

La solución que se ha dado a este problema es disponer de un catálogo de dispositivos domóticos predefinidos para cada una de las tecnologías existentes (por lo menos de las más importantes). En este catálogo se indicará para cada dispositivo que funcionalidades lo compone. Por tanto existirá un modelo de componentes a nivel PIM y un segundo modelo de componentes, fruto de un refinamiento del primero, que se situará a nivel PSM.

Por este motivo se hace indispensable tener catalogados los distintos dispositivos que se pueden encontrar en las diferentes tecnologías teniendo en cuenta las funcionalidades que ofrecen. Y dado que las unidades funcionales son representadas en el modelo de componentes como un componente simple, la forma más sencilla de realizar esta operación de catalogación es mediante un refinamiento del modelo de componentes.

4.3.1 Metamodelo de Componentes

Una arquitectura de componentes se caracteriza por tener como elementos arquitectónicos componentes y conectores. Para la definición de estos componentes y conectores se han definido una serie de *Eclass* y se han tenido en cuenta las posibles relaciones de inclusión o transformación que serán necesarias. En las Figuras 13 y 14 se puede observar dicho metamodelo en formato *TreeEditor* y *EDiagram* respectivamente.

Figura 13: Metamodelo de componentes en formato TreeEditor.

Figura 14: Metamodelo de componentes en formato EDiagrama .

4.3.2 Catálogo de Componentes

Como se ha comentado, para poder realizar el refinamiento de un modelo de componentes es necesario tener un modelo donde se recojan en forma de catálogo aquellos componentes complejos o incluso componentes con una funcionalidad simple. A continuación se muestra que información recoge cada uno de los elementos del catálogo.

ComplexComponent: Un componente complejo representa un dispositivo domótico para una determinada plataforma. Este componente está formado por uno o varios

componentes simples, pudiendo darse el caso que los componentes simples que lo compone sean entre ellos del mismo tipo. Los atributos más importantes de un *ComplexComponent* son:

- **Name (EString):** Indica el nombre del componente y funciona como atributo identificador. Este nombre debe coincidir con el nombre del dispositivo real.
- **Tag1 (EString):** es un campo auxiliar que puede utilizarse para añadir una descripción más detallada del componente.

SimpleComponent: Un componente simple representa una funcionalidad interna de un dispositivo. Dentro del catálogo sólo se pueden encontrar integrados en un componente complejo. Los atributos más importantes de un *SimpleComponent* son:

- **Name (EString):** Indica el nombre del componente, este nombre coincide con el nombre que reciben las definiciones de unidades funcionales del DSL. En cierto modo identifica el tipo de funcionalidad que ofrece este componente.
- **Owned:** Muestra el *ComplexComponent* donde está encapsulado este *SimpleComponent*
- **Tag1 (EString):** Indica el índice del SimpleComponent. Este campo toma la función junto con el nombre de identificador, dado que un mismo ComplexComponent puede contener varios componentes del mismo tipo.

Port: Todo componente, ya sea simple o complejo, va a tener una serie de puertos. Los puertos representan los puntos de iteración del componente con el resto del sistema. Los atributos más importantes de un *Port* son:

- **Name (EString):** Indica el nombre del puerto, este nombre sigue una nomenclatura del tipo “port_NombreComponente_Indice”.
- **Provided:** Indica la interfaz provista.
- **Required:** Indica la interfaz requerida.

PortLink: : Este elemento permite crear un vínculo entre los componentes o mejor dicho entre los puertos de los componentes. Estos enlaces pueden ser del tipo “delegation”, este tipo de enlace conecta un puerto definido en un componente interno con un puerto compatible definido en su componente complejo contenedor. Los enlaces no delegation conectan dos puertos de dos componentes definidos al mismo nivel.

- **delegation (EBoolean):** Indica si es un portlink del tipo delegation o no.
- **endA (Port):** Indica el puerto origen.
- **endB (Port):** Indica el puerto destino.

4.3.3 Refinamiento de Componentes

El refinamiento se define en términos de MDA como la adición de los detalles necesarios para pasar de un modelo abstracto (por ejemplo un PIM) a un modelo más detallado (que puede ser un PSM) que en general estará más cerca de una plataforma de implementación. En el caso del dominio domótico los detalles que se han añadido han sido las agrupaciones de las distintas funcionalidades que conforman el sistema a desarrollar según la disponibilidad de dispositivos de la plataforma final seleccionada.

Para realizar el proceso de refinamiento se necesita:

- Disponer de un catalogo de dispositivos representados como componentes compuestos. Estos componentes compuestos indicarán de que funcionalidades se componen (representadas como componentes simples) y cómo interactúan entre ellas.
- Una transformación de refinamiento donde los modelos origen y destino son conformes al mismo metamodelo de componentes.
 - Se utilizarán dos modelos de entrada, 1) el modelo de componentes simples directamente obtenido de la transformación del modelo de aplicación DSL y 2) el modelo de componentes compuestos que representa el catalogo de dispositivos.
 - El modelo destino se corresponde con un modelo de componentes que modela una aplicación domótica estando ahora representados los dispositivos concretos a utilizar en la implantación final.
- Buscar mediante pattern matching coincidencias entre los componentes simples que componen cada uno de los dispositivos catalogados como componentes compuestos y los componentes simples del modelo de aplicación. (véase Figura 15) dando lugar al modelo de componentes compuestos de la aplicación.
- El modelo de componentes obtenido ya ha adquirido una dependencia de la plataforma y está preparado para ser transformado a un modelo de plataforma final.

Figura 15: Esquema del proceso de refinamiento de componentes.

Las trazas que todos estos artefactos generan durante el refinamiento se almacenan en un modelo de trazabilidad para necesidades futuras.

4.4 Plataforma de Implementación

Finalmente el proceso termina con la generación de código y la implementación real del sistema domótico. Para ello se debe seleccionar la plataforma específica de implementación (esta decisión se ha tomado al realizar el refinamiento de componentes). Dada la experiencia del grupo de investigación en la tecnología KNX en una primera aproximación se ha utilizado KNX como plataforma final, aunque la metodología propuesta proporciona los mecanismos para realizar el desarrollo en cualquier otra plataforma. Para ello se proporcionara para cada plataforma / tecnología un metamodelo (en la Figura 16 se muestra el metamodelo para KNX) que recoja los conceptos específicos de dicha plataforma y posteriormente se realizará una transformación M2T obteniendo el código del sistema.

Figura 16: Metamodelo para la plataforma KNX.

Para el nivel específico de la plataforma EIB/KNX se ha apostado por el aprovechamiento de las herramientas disponibles en lugar de construir un conjunto de herramientas a partir de cero.

En la actualidad el punto de partida para un desarrollador tradicional sería la creación del proyecto mediante la iteración manual con la herramienta ETS (acrónimo del inglés *Engineering Tool Software*). Con la nueva metodología que se propone en este trabajo de Tesis, esa iteración desaparece y la herramienta se convierte en un soporte para la ejecución del código obtenido.

El programa ETS es la única herramienta software independiente del fabricante para diseñar y configurar instalaciones inteligentes para el control de casas y edificios hechas con el sistema KNX. La Herramienta *Software* para Ingeniería ETS3 se estructura por

tanto de manera flexible, extensible y modular para poder facilitar futuras ampliaciones de la tecnología EIB/KNX. Asimismo, se ofrece al usuario una amplia ayuda en línea que facilita toda la información necesaria en este contexto. La herramienta ETS además puede ampliar su funcionalidad mediante una serie de plugins disponibles.

El plugin IT Tool de la herramienta ETS proporciona un entorno de desarrollo para escribir el código de programación de los sistemas domóticos. Este código se escribe en el lenguaje VBScript y se organiza en una serie de ficheros llamados macros. Estas macros son ejecutadas por la herramienta ETS obteniendo así la configuración final que se implementa en cada uno de los dispositivos.

Todas las operaciones realizadas por una macro pueden hacerse también “a mano” con la herramienta ETS. Sin embargo, las macros son más rápidas y cometen menos errores. De esta manera, la principal ventaja del uso de macros es que ahorran tiempo y permite automatizar la creación de un proyecto domótico.

La metodología propuesta para esta fase se resume en la Figura 17, donde se puede apreciar como se realiza una transformación de los modelos KNX a una macro ejecutable en ETS que genere el proyecto. Esta transformación se realiza con la herramienta de transformación M2T JET.

Figura 17: Esquema del proceso de generación de código.

4.5 Trazabilidad

En esta Tesis la trazabilidad se gestiona de forma transversal al resto de etapas o niveles de la metodología, por lo tanto la única forma de poder integrarla en una propuesta MDE es definiendo un metamodelo en el que dejar constancia de las relaciones relevantes entre los distintos artefactos involucrados. En la Figura 18 se muestra el metamodelo de trazabilidad adoptado, este contiene un elemento llamado Link que contiene dos referencias a cualquier *ModelElement*, es decir a cualquier artefacto definido en un metamodelo. Esto permite aplicar este enfoque de trazabilidad en cualquier dominio, en cualquier nivel de abstracción y con cualquier artefacto origen o destino. El *CompositeLink* permite definir distintos niveles de granularidad para agrupar Links en otros más complejos, según el propósito global al que están refiriéndose. El atributo *linkType* permite categorizar las relaciones existentes y distinguir en qué nivel del proceso de desarrollo se ubica la traza.

El proceso considera un modelo de trazabilidad entre cada dos niveles de abstracción consecutivos. Por ejemplo, para el caso particular de la trazabilidad entre requisitos domóticos y el modelado correspondiente con el DSL, el elemento Link del metamodelo es definido por:

- Un *ModelElement* (origen) que referencie a un elemento del metamodelo de requisitos domóticos.
- Un *ModelElement* (destino) que referencie el elemento o elementos del DSL relacionados con dicho requisito.

Figura 18: Metamodelo de trazabilidad propuesto.

La Figura 19 muestra un ejemplo de elementos trazados entre dos niveles de abstracción consecutivos (de requisitos a diseño, del diseño a elementos del lenguaje de programación, etc.). Puede observarse cómo es posible integrar las relaciones uno a muchos dentro de un mismo Link. Al mismo tiempo, un target puede ser destino de diversas trazas con distintos orígenes.

Los modelos de trazabilidad se crean durante el proceso de transformación. El primer modelo de trazabilidad (entre requisitos domóticos y el modelo DSL) es creado manualmente. Desde el DSL hasta el código final generado, todas las trazas se obtienen automáticamente como parte del proceso de transformación de modelo origen a destino. Dicho de otra forma, al mismo tiempo que se ejecutan las transformaciones relativas a la obtención de modelos más cercanos a la plataforma de ejecución se anota en el repositorio de trazabilidad el vínculo existente entre los elementos involucrados. De esta

forma, para disponer de esta facilidad de traza deben extenderse las reglas de transformación con una parte dedicada a anotar la traza en el metamodelo definido.

Figura 19: Ejemplos de trazas que pueden representarse con el metamodelo de trazabilidad propuesto.

Para crear dichas trazas se han extendido las reglas de transformación, añadiendo un nuevo elemento *TraceLink* en las transformaciones de grafos. Cada elemento *TraceLink* de la regla de transformación recoge la información necesaria (*name*, *description*) para las instancias de tipo *Link* del metamodelo de trazabilidad presentado en la Figure 18. También se almacenan las correspondencias entre *source* y *target*. De esta forma, quedan recogidas en el “*Traceability Model #2*” las trazas existentes entre los elementos del nivel DSL y los del modelo de componentes PIM.

También se ha incluido la creación de las trazas en las transformaciones ATL realizadas. Para ello se han incorporado las siguientes *called rule* que son llamadas cada vez que se transforma un artefacto.

```
rule CrearTrazasComposite(n:String) {
  to
    u:C!CompositeLink( )
  do{
    u.description<-n;
    u.name<-n;
    thisModule.trazaRoot.hasTraceLink<-u;
    thisModule.CompositeLink<-u;
  }
}

rule CrearTrazasHijas(n:String,s:MOF!EObject,t:MOF!EObject) {
  to
    u:C!Link( )
  do{
    u.source<-s;
    u.target<-t;
    u.description<-n;
    u.name<-n;
    thisModule.CompositeLink.child<-u;
  }
}
```

```
rule CrearTrazasSimples(n:String,s:MOF!EObject,t:MOF!EObject){
  to
    u:C!Link(    )
    do{
      u.source<-s;
      u.target<-t;
      u.description<-n;
      u.name<-n;
      thisModule.trazaRoot.hasTraceLink<-u;
    }
}
```

4.5.1 El informe de trazabilidad

Lo realmente interesante del enfoque de trazabilidad implementado es que una vez se ha generado el modelo de trazabilidad es posible obtener un informe detallado de los elementos trazados utilizando la herramienta RTT (*Requirements-Trace-Tool*) desarrollada (véase Figura 20). La herramienta, desarrollada utilizando JET, genera el informe de trazabilidad en formato HTML aplicando una transformación M2T a los modelos de trazabilidad existentes.

Este informe permite a los diseñadores de sistemas domóticos analizar cómo un requisito ha sido tenido en cuenta en el proceso de generación automática de código e información de la solución adoptada. La Figura 20 muestra una captura de pantalla del uso de la herramienta RTT, donde se pueden apreciar los elementos almacenados en el “*Traceability Model #1*”. Para cada una de las trazas se detalla su nombre y descripción, así como la información relativa al *source* y el *target* de los elementos trazados. La primera fila del informe muestra un *CompositeLink* que agrupa toda la información relativa a los *Links* involucrados en la solución usando el DSL para el requisito de iluminación. En este caso, los Links representan las trazas entre los requisitos Req#2 y Req#3 que tienen asociado como *target* los elementos del DSL indicados (unidades funcionales y enlaces entre ellas). La Figura 21 muestra el informe de trazabilidad correspondiente al “*Traceability Model #2*”, generado automáticamente como parte del proceso de transformación de la especificación usando el DSL al modelo de componentes. Para simplificar la presentación, únicamente se muestran las correspondencias entre dos unidades funcionales (DMI-1 y PB-1) y los elementos generados tras la ejecución de la transformación al DSL (componentes, puertos e interfaces). Además, se incluye la traza relativa a la transformación realizada para el enlace entre las unidades funcionales llamado C1 que tiene como destino un Binding llamado Link PB-1 to DMI-1. Los modelos “*Traceability Model #3*” se generan en el proceso de ejecución de las transformaciones M2M desde el modelo de componentes al modelo correspondiente a la plataforma KNX/EIB.

Con todo esto, el informe de trazabilidad permite al usuario distinguir qué requisitos han sido considerados, qué elementos arquitectónicos dan soporte a cada uno de ellos, y por último, en qué dispositivos físicos de la plataforma se han recogido las implementaciones.

4. Metodología propuesta

Traceability Model #1						
Trace Link		Source Link		Target Link		
Name	Description	Metamodel	Model Element	Metamodel	Model Element	Composite?
Requisito Luz	Transformación de requisitos de luz a los fragmentos correspondientes de la aplicación domótica (aplicación DSL)					✓
Req3 to DSL	Transformación del requisito 3 al DSL	Requirement	Requirement:hasRequirements - name => Encendido/Apagado automático luz - description => Encendido/Apagado de un punto de luz de forma automática por detección de presencia - validationProcedure => Entrar en una habitación y comprobar que se enciende automáticamente la luz - id => REQ3	MetaDomo	MetaDomo:hasFUnitInstance - FUnitInstanceName => PIR3-2 - FUnitInstanceRef =>	Child of: Requisito Luz
				MetaDomo	MetaDomo:hasFUnitLink - FUnitLinkName => C3 - canal => true	Child of: Requisito Luz
				MetaDomo	MetaDomo:hasFUnitInstance - FUnitInstanceName => SW7-1 - FUnitInstanceRef =>	Child of: Requisito Luz
				MetaDomo	MetaDomo:hasFUnitLink - FUnitLinkName => L7	Child of: Requisito Luz
				MetaDomo	MetaDomo:hasFUnitInstance - FUnitInstanceName => TM-1	Child of: Requisito Luz
				MetaDomo	MetaDomo:hasFUnitLink - FUnitLinkName => L8	Child of: Requisito Luz
				MetaDomo	MetaDomo:hasFUnitLink - FUnitLinkName => L9	Child of: Requisito Luz
				MetaDomo	MetaDomo:hasFUnitLink - FUnitLinkName => L10	Child of: Requisito Luz
Req2 to DSL	Transformación del requisito 2 a DSL	Requirement	Requirement:hasRequirements - name => Regulación Luz - description => Regulación de la intensidad luminica de un punto de luz - validationProcedure => Comprobar que al accionar la regulación de intensidad luminica se incrementa/decrementa de manera correcta - id => Req2	MetaDomo	MetaDomo:hasFUnitInstance - FUnitInstanceName => PB-1	Child of: Requisito Luz
				MetaDomo	MetaDomo:hasFUnitLink - FUnitLinkName => C1 - canal => true	Child of: Requisito Luz
				MetaDomo	MetaDomo:hasFUnitInstance - FUnitInstanceName => DMI-1	Child of: Requisito Luz

Figura 20: Informe de trazabilidad Traceability Model #1.

Traceability Model #2						
Trace Link		Source Link		Target Link		
Name	Description	Metamodel	Model Element	Metamodel	Model Element	Composite?
Req3 to V3CM						✓
PB-1 to V3CM	Transformation PB-1 to V3CM	MetaDomo	MetaDomo:hasFUnitInstance - FUnitInstanceName => PB-1	v3cm	simpleComponent - name => PB-1 (Pushbutton)	Child of: Req3 to V3CM
				v3cm	component:Port - name => P_PB_DMI	Child of: Req3 to V3CM
				v3cm	interface - name => I_inSwitch	Child of: Req3 to V3CM
DMI-1 to V3CM	Transformation DMI to V3CM	MetaDomo	MetaDomo:hasFUnitInstance - FUnitInstanceName => DMI-1	v3cm	simpleComponent - name => DMI-1 (Dimming In)	Child of: Req3 to V3CM
				v3cm	component:Port - name => P_DMI_PB	Child of: Req3 to V3CM
				v3cm	interface - name => I_inSwitch	Child of: Req3 to V3CM
C1 to V3CM	Transformation C1 to V3CM	MetaDomo	MetaDomo:hasFUnitLink - FUnitLinkName => C1 - canal => true	v3cm	component:Binding - name => Assembly Link PB-1 to DMI-1	Child of: Req3 to V3CM

Figura 21: Informe de trazabilidad Traceability Model #2.

5. RESÚMENES COMPENDIO DE ARTICULOS

En este apartado se aportan los resúmenes en castellano de los cuatro artículos que forman el compendio de publicaciones de esta Tesis.

- **Habitation: A Domain Language for Home Automation** (Jimenez et al. 2009a). Este artículo ha sido publicado en la revista “*IEEE Software*” en el año 2009. Esta revista esta indexada como JCR (Journal Citation Report) por el ISI, con un factor de impacto 2.039, encontrándose en el primer cuartil de su categoría. La principal aportación de este artículo es un DSL (Domain Specific Language – Lenguaje Específico de Dominio) específico para el dominio domótico que permite modelar una aplicación domótica de una forma fácil, intuitiva y totalmente independiente de la plataforma.
- **A tool for facilitating the teaching of smart home application** (Jiménez et al. s. f.). Este artículo ha sido publicado en la revista “*Computer Applications in Engineering Education*” en el año 2011. Esta revista esta indexada como JCR (Journal Citation Report) por el ISI, con un factor de impacto 0.321 (tercer cuartil de su categoría). La principal aportación de este artículo es una herramienta para la docencia de los sistemas domóticos basada en el DSL presentado en (Jimenez et al. 2009a), permitiendo de este modo validar la funcionalidad del DSL en una aplicación real.
- **Introducing Safety Requirements Traceability Support in Model- Driven Development o Robotic Applications** (Sanchez et al. 2011). Este artículo ha sido publicado en la revista “*IEEE Transactions on Computers*” en el año 2011. Esta revista esta indexada como JCR (Journal Citation Report) por el ISI, con un factor de impacto 1.608 (segundo cuartil de su categoría). Este artículo no trata el dominio domótico en sí, sino que la aportación a la Tesis es el enfoque de trazabilidad y la herramienta para la gestión de los modelos de trazabilidad que en él se presenta.
- **A framework for developing home automation systems: From requirements to code** (Sánchez et al. 2011). Este artículo ha sido publicado en la revista “*Journal of Systems and Software*” en el año 2011. Esta revista esta indexada como JCR (Journal Citation Report) por el ISI, con un factor de impacto 1.293 (segundo cuartil de su categoría). Este artículo es el más completo y el que mayor aportación realiza a la Tesis. En él se presenta un framework que da soporte completo al desarrollo de aplicaciones domóticas, empezando desde la captura de requisitos hasta la generación de código. También hay que indicar que en este artículo se integran conceptos, enfoques y herramientas presentadas en los artículos anteriores, siendo el caso por ejemplo del DSL o el enfoque de trazabilidad.

Habitation: A Domain-Specific Language for Home Automation

En este artículo se presenta un lenguaje específico de dominio (DSL) que constituye un elemento fundamental de una nueva metodología de desarrollo basada en el enfoque dirigido por modelos (MDE) como alternativa para resolver los problemas encontrados en el desarrollo tradicional de software para sistemas domóticos.

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 la arquitectura dirigida por modelos (MDA, *Model Driven Architecture*) [2] que utilizan lenguajes de modelado, como UML, [5][6] poco intuitivos y alejados del dominio domótico. Otras propuestas estudiadas se corresponden con herramientas comerciales dependientes de la plataforma, de las que las más conocidas son ETS (*Engineering Tool Software*) y LonMaker, específicas de las plataformas EIB/KNX [7] y Lonworks[8] respectivamente

El problema es que los sistemas domóticos 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 es mínima.

Los DSL aportan conceptos de un dominio de aplicación y permiten identificar primitivas de alto nivel al recoger conocimiento de expertos en dicho dominio. En este nuevo contexto, estos lenguajes facilitan el trabajo en las primeras etapas de diseño. Además, el enfoque MDE ayuda a reducir el coste de desarrollo de los DSL, dado que en MDE el objetivo principal es construir software a partir de modelos, de manera que los DSL toman un papel importante como lenguajes de modelado que permiten describir el sistema mediante modelos gráficos de una forma fácil e intuitiva.

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 incrementan 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 las viviendas, dotándolas de una cierta "*inteligencia*" con el fin de mejorar la vida cotidiana de las personas [4].

La utilización del DSL propuesto en este documento proporciona al usuario (que ahora no tiene que ser experto) un entorno gráfico con un alto nivel de abstracción donde definir los requisitos del sistema con conceptos propios del dominio domótico, quedando descrita toda la lógica de negocio desde una perspectiva independiente de computación y de la plataforma. De este modo no se hace necesario comenzar el desarrollo analizando el software desde una perspectiva tecnológica.

El beneficio del uso de DSLs en el dominio de la domótica es por tanto doble: (i) Se ha conseguido separar en dos vistas complementarias el catálogo de dispositivos (presentes o futuros) y el diseño de nuevas aplicaciones; (ii) al integrarse esta propuesta en un marco de desarrollo dirigido por modelos (dicha integración se aplica en [17]), se obtiene una infraestructura conceptual más adecuada para beneficiarse de las posibilidades relativas a: generación automática de código, verificación y validación de modelos, trazabilidad de requisitos, etc.

Como validación de la propuesta del DSL presentado en este artículo, se realizó una serie de experimentos de los cuales se obtuvieron unos resultados muy positivos en cuanto a la usabilidad del DSL, aunque entendemos que efectivamente medir el beneficio del DSL no pasa sólo por hablar de usabilidad sino que hay otras características como la mantenibilidad, funcionalidad, fiabilidad, eficiencia y portabilidad que no pueden olvidarse (no se incorpora el experimento en el resumen dado la extensión del mismo, pero se encuentra totalmente detallado en el artículo original [18]).

A continuación se presenta el DSL HABITATION (development of Home Automation ApplicTIons using a mOdel driveN approach en detalle, principal aportación de este artículo.

UN DSL PARA EL DESARROLLO DE APLICACIONES DOMOTICAS

El DSL HABITATION que se presenta en este trabajo facilita la captura de requisitos propios de un sistema domótico de una forma visual e intuitiva. De esta manera, los usuarios ven facilitada la posibilidad de expresar y entender su conocimiento y experiencia en el dominio. Por esta razón, la primera premisa que se tuvo en cuenta fue la de disponer de una riqueza semántica importante para la visualización del conocimiento, pero a la vez concisa y común a las distintas plataformas. Antes de entrar en detalles es necesario exponer los principales conceptos con los que se trabaja en el dominio domótico y que deben tenerse en cuenta a la hora de crear el DSL.

En cualquier sistema domótico existe una serie de elementos (que denominamos "*Unidades Funcionales*") que aparecen en todas las tecnologías y estándares domóticos. Estas tecnologías y estándares se diferencian en la arquitectura, protocolos utilizados, dispositivos o módulos disponibles, pero son muy similares en cuanto a la funcionalidad que ofrecen. Con el fin de promover la reutilización de estas unidades funcionales y evitar tener que definir múltiples 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 reutilizables, de manera que una vez definido dicho catálogo éste se pueda utilizar en cualquier aplicación y sólo sea necesario instanciar elementos del mismo. 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, de manera que se ha creado un *Catálogo de Servicios* con unas *Definiciones de Servicios* que puedan ser reutilizadas en cualquier unidad funcional.

Por esta razón es conveniente distinguir dos vistas del DSL, una para el desarrollo de aplicaciones con un desarrollador de aplicaciones como usuario (conoce el dominio

pero no tiene que ser experto) y una segunda vista para desarrollar y realizar posibles actualizaciones del catálogo, donde el usuario deber ser un experto en el dominio.

Vista Catálogo del DSL

El DSL permite al experto en el dominio modelar el catálogo de unidades funcionales y servicios. Este catálogo será utilizado posteriormente para el desarrollo de aplicaciones domóticas, sin que sea necesario definir cada una de las unidades funcionales y sus servicios cada vez que se desarrolle una nueva aplicación.

Las primitivas disponibles para modelar un catálogo se describen en la Tabla y estarán disponibles en la paleta de la herramienta desarrollada.

La actualización y ampliación de dicho catálogo se realizará mediante la incorporación de nuevas definiciones de unidades funcionales dentro de las categorías existentes.

Vista Aplicación del DSL

Esta vista del DSL será utilizada por el desarrollador, encargado de definir/diseñar nuevas aplicaciones y sin tener que ser un experto en el dominio domótico. En este sentido, y gracias a la existencia del catálogo, la especificación de una aplicación domótica se realiza mediante:

- La instanciación de unidades funcionales que vienen definidas en el catálogo. Estas instancias de unidades funcionales deben ser configuradas añadiendo los valores necesarios a sus parámetros. Por ejemplo, para una unidad funcional definida como DMI que tiene un parámetro *DMILongPushTime* del tipo numérico se deberá indicar el valor que debe tomar dicho parámetro. Se podría dar un valor para que pasados 3 segundos de pulsación detecte una pulsación larga y empiece a regular, en caso contrario realizará una conmutación.
- Los enlaces entre unidades funcionales. Mediante enlaces se puede indicar la forma en la que las unidades funcionales van a interactuar con el resto del sistema. Al realizar un enlace se indican los servicios que se interconectan. Estos enlaces pueden comportarse como enlaces de tipo *canal* en el caso de que una de las unidades funcionales involucradas sea pasiva de manera que se modela una conexión hardware o bien como un enlace normal.
- Escenas. Se puede configurar la ejecución de varios servicios de unidades funcionales de forma secuencial con una única acción. Por ejemplo se puede crear una escena "Presentación" que se activa desde un pulsador y ejecuta una serie de servicios prestablecidos con antelación, por ejemplo bajar persiana, bajar intensidad de las luces y bajar pantalla de proyección.

Las primitivas disponibles en este DSL son las que se muestran en la Tabla 1 y estarán disponibles en las paletas de la herramienta desarrollada (véase Figura 2).

Representación gráfica y descripción de las primitivas de la vista Catálogo
Categoría: Especialización de un elemento del catálogo.
Enlaces entre categorías (verde) y enlaces entre unidades funcionales y categorías (azul).
Definición de Unidad Funcional: es el mínimo elemento en el que puede dividirse un dispositivo domótico. Incluye un icono, un nombre y los servicios provistos e implementados..
Servicio: Una definición de servicio tiene una firma que incluye el nombre del servicio y sus argumentos. El servicio indica si es provisto o requerido (flecha izquierda o derecha) e indica si es un servicio hardware o no (flecha roja o azul). Un catálogo de servicios se usa para permitir la reutilización de los servicios.
Definición de parámetro: indica el parámetro de una unidad funcional.
Definición de escena: es una especialización de unidad funcional.
Representación gráfica y descripción de las primitivas de la vista Aplicación
Instancia de unidad funcional: incluye los parámetros y sus valores.
Enlace entre unidades funcionales. Un enlace del tipo canal se muestra en rojo, el resto de enlaces se muestran con una línea discontinua verde. Estos enlaces tienen tres etiquetas con los nombres de los servicios involucrados en dicho enlace.
Escena: contiene los pasos de escena que se ejecutarán. Un paso de escena muestra el servicio y el icono de la unidad funcional que participa en dicho paso de escena..

Tabla 1: Primitivas Vista Catálogo y Aplicación del DSL.

HERRAMIENTA DSL Y ENTORNO DE TRABAJO

Como se ha detallado a lo largo del documento, el propósito del DSL presentado es proporcionar una herramienta gráfica que permita a los usuarios capturar los requisitos específicos de una aplicación domótica de una forma fácil e intuitiva, utilizando conceptos del dominio domótico. Para ello se han utilizado unas herramientas de modelado integradas todas ellas en un mismo entorno de trabajo.

Todas estas herramientas se encuentran integradas en Eclipse [10], un entorno de desarrollo de libre distribución y código abierto que proporciona un marco de trabajo donde toda la funcionalidad necesaria puede obtenerse mediante plugins, consiguiendo así un entorno totalmente extensible. 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 emplea el plugin de modelado EMF (*Eclipse Modeling Framework*) [11] que proporciona una infraestructura de modelado con la que crear editores de modelos. EMF trabaja con un subconjunto de MOF[12] llamado EMOF (*Essential MOF*) que permite crear, manipular y almacenar modelos y metamodelos de una forma más simple. Lo más importante de todo es que EMF suministra las bases para la interoperabilidad con otras herramientas y aplicaciones basadas en MOF. Sobre la infraestructura ofrecida por EMF puede utilizarse la herramienta de generación de editores gráficos GMF (*Graphical Model Framework*) [13] que surge con el objetivo de generar automáticamente editores gráficos a partir de modelos.

El DSL se ejecuta en un entorno Eclipse, por lo que mantiene una estructura visual muy parecida con el resto de herramientas de Eclipse. En la Figura 1 se pueden observar tres zonas principales del DSL:

- Área de dibujo: zona de trabajo donde se arrastran las figuras desde la paleta para construir un modelo gráfico concreto, tanto para la vista de Catálogo como de Aplicaciones. También existe la opción de instanciar elementos desde un menú flotante en lugar de utilizar la paleta.
- Paleta gráfica: contiene las primitivas que se pueden arrastrar a la zona de dibujo. En la Figura se puede apreciar las paletas de la vista Catálogo y Aplicación respectivamente.
- 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 1: Vista Aplicación del DSL .

(a) Vista Catálogo.

(b) Vista Aplicación

Figura 2: Paletas DSL.

En el artículo original se presenta un caso de estudio donde se demuestra la funcionalidad del DSL en una aplicación real. También pueden consultar un vídeo con una demostración del caso de estudio en la siguiente url <http://hdl.handle.net/10317/854>.

REFERENCIAS

- [1] Bran Selic, "The Pragmatics of Model-Driven Development". IEEE Software, 20(5):46–51, September/October 2003.
- [2] J. Miller, J. Mukerji, "MDA Guide Version 1.0.1.", OMG, 2003.
- [3] J. Mernik M., Lämmel R., "Special issue on domain-specific languages, Part II.", Journal for Computing and Information Technology, 10(1), 200 Miller, J.
- [4] J.L. Ryan, "Home Automation", Electronics & Communication Engineering Journal, Volume: 1 Issue:4, July-Aug. 1989, pp 185-192
- [5] Voelter M., Groher I., "Product line implementation using aspect-oriented and model-driven software development." SPLC 2007: 11th International Software Product Line Conference, Proceedings. (2007):233-242.
- [6] Muñoz J., Pelechano V., Cetina C., "Implementing a Pervasive Meetings Room: A Model Driven Approach.", III International Workshop on Ubiquitous Computing (IWUC 2006), Paphos (Cyprus), May 2006
- [7] KNX standard disponible en <http://www.knx.org/>
- [8] LonWorks standard disponible en <http://www.lonmark.org/>
- [9] Object Management Group (OMG) disponible en <http://www.omg.org/>
- [10] Web oficial de la Plataforma Eclipse, disponible en <http://www.eclipse.org>

- [11] Web oficial del proyecto EMF (Eclipse Modeling Framework) disponible en <http://www.eclipse.org/emf/>
- [12] Meta-Object Facility (MOF) Specification v2.0, ptc/04-10-15. Object Management Group (OMG), www.omg.org/technology/documents/modeling_spec_catalog.htm
- [13] Web oficial del proyecto GMF (Graphical Modeling Framework) disponible en <http://www.eclipse.org/gmf/>
- [14] ISO/IEC 9126 International Organization Tichy, W. "Hints for reviewing empirical work in softwareengineering. Empirical Software Engineering" 5, 309–312. 2000
- [15] Wohlin, C., Runeson, P. "Experimentation in Software Engineering: An Introduction". Kluwer Academic Publisher, Boston, MA. 2000
- [16] A. Seffah, M. Donyae, R.B. Kline,"Usability measurement and metrics:A consolidated model". Software Qual J (2006) 14: 159–178

A Tool for Facilitating the Teaching of Smart Home Applications

En este artículo se presenta una nueva herramienta para mejorar los métodos de enseñanza del desarrollo de sistemas domóticos. La principal aportación de esta herramienta es que permite desarrollar estos sistemas de forma totalmente independiente de la plataforma, aportando además un entorno gráfico en el que trabajar. Los docentes pueden utilizar por tanto esta herramienta para introducir el desarrollo de aplicaciones domóticas y para demostrar los conceptos básicos en este ámbito, centrándose en las propiedades y conceptos globales de los sistemas domóticos en lugar de los específicos de cada plataforma o tecnología, como EIB/KNX, Lonworks o X10 [1],[2].

Si la comparamos otras herramientas para domótica, las principales ventajas de esta propuesta son: (1) el uso de una nivel de abstracción mayor y totalmente independiente de la plataforma domótica del despliegue final, (2) la utilización de conceptos como por ejemplo el de “*unidad funcional*” en lugar de dispositivos para lograr la independencia de la plataforma, (3) estas unidades están agrupadas por categorías dentro de un catálogo que es ampliable, por lo que la herramienta nunca queda desactualizada y (4) cuenta con un lenguaje gráfico específico para el dominio domótico que permite modelar una aplicación domótica de una forma fácil e intuitiva. Gracias a estas características, la herramienta puede ser adoptada como un complemento a las clases teóricas.

Por otro lado, los estudiantes pueden utilizar la herramienta como un medio autónomo de aprendizaje. Desde el punto de vista de los estudiantes, el desarrollo de un nuevo sistema comienza con el uso de un lenguaje gráfico específico para domótica, lo que permite prescindir tener que aprender el uso de herramientas concretas para cada plataforma y obtener fácilmente un modelo de la aplicación en un tiempo considerablemente menor. Además, como la herramienta está a la completa disponibilidad del estudiante, éste puede utilizarla en casa para desarrollar parte de su trabajo (lo que promueve el auto estudio y trabajo en casa como se propone en el proceso de Bolonia). Esto no sería posible con herramientas comerciales como ETS (Software Engineering Tool para KNX) o LonMaker (la herramienta de desarrollo para Lonworks) que requieren comprar las respectivas licencias que suponen un costes económicamente elevado. Este nuevo método pretende motivar a los estudiantes y ayudarles a aprender de una forma más rápida y sencilla, obteniendo por tanto mejores calificaciones. Estas calificaciones más favorables podrían estar relacionados con el hecho de que la herramienta permite a los estudiantes obtener una comprensión más profunda de los conceptos de HA y no sólo de programación y asuntos dependientes de la plataforma.

Este artículo por tanto se aporta una herramienta y un entorno de trabajo donde se valida una vez más la aplicación real del DSL HABITATION. A parte de la descripción de dicha herramienta, también se define la metodología a seguir por profesores y alumnos

DETALLE DE LA HERRAMIENTA

El objetivo de este trabajo es proporcionar a los estudiantes (y a cualquier usuario) una herramienta flexible e intuitiva para el desarrollo de aplicaciones domóticas manteniendo la independencia de la plataforma final de implementación. Gracias a ella los alumnos asimilan mejor los conceptos generales aprendidos en el campo de la domótica dado que pueden aplicar los conceptos directamente sin necesidad de aprender otros nuevos dependientes de las distintas plataformas (KNX, X10, Lonworks, Zigbee, ...). Por otro lado, deja de ser necesario el uso de herramientas existentes de complicado manejo y de alto coste económico.

Esta herramienta se fundamenta en la utilización conjunta de una metodología MDE (Model Driven Engineering) [3] junto al lenguaje específico de dominio (DSL) [4] para domótica. En concreto hace uso del lenguaje desarrollado en el seno del grupo de investigación DSIE de la UPCT llamado HABITATION [5]. Para su desarrollo se ha usado la arquitectura dirigida por modelos (MDA) propuesta por el Object Management Group (OMG). Esta arquitectura está organizada en tres capas o niveles para el desarrollo del software: (1) nivel de modelos independientes de la computación (CIM) donde se representa la sintaxis y parte de la semántica para definir el DSL, (2) nivel de modelos independientes de la plataforma (PIM) construido como un modelo de componentes y (3) un nivel de modelos específicos de la plataforma domótica (PSM). El desarrollador tan solo tiene que definir la aplicación domótica con el DSL, automáticamente se realizarán una serie de transformaciones entre los modelos de los distintos niveles de abstracción hasta alcanzar el modelo específico de la plataforma seleccionada y finalmente este modelo es transformado de nuevo obteniendo el código ejecutable del sistema domótico desarrollado. Todo este proceso es totalmente transparente al usuario.

La parte visible de la herramienta, con la que interaccionan los estudiantes, se corresponde con el editor de modelos gráficos proporcionado por el DSL del nivel CIM. Por tanto, en este documento nos vamos a centrar en dicha parte dado que el resto de la metodología se ejecuta de forma transparente al alumno.

La primera premisa que se tuvo en cuenta a la hora de crear la herramienta fue la de disponer de una riqueza semántica importante para la visualización del conocimiento, pero a la vez concisa y común a las distintas plataformas. Por ello ha sido necesario identificar muy bien los principales conceptos del dominio domótico para poder crear el DSL de la herramienta.

En cualquier sistema domótico existen una serie de elementos, que denominamos Unidades Funcionales (“*FUnit*”) que aparecen en todas las tecnologías y estándares domóticos. Estas tecnologías y estándares se diferencian en la arquitectura, protocolos utilizados o módulos disponibles, pero son iguales en cuanto funcionalidad, por esto es mejor trabajar a nivel de unidad funcional en lugar de trabajar directamente con dispositivos, así aseguramos la independencia de la herramienta respecto a cualquier plataforma. Estas unidades funcionales a su vez disponen de unos “*Services*” gracias a los cuales las unidades funcionales podrán interaccionar entre ellas. Con el fin de promover la reutilización de estas unidades funcionales y evitar tener que definir múltiples veces la misma unidad para cada aplicación (incluso varias veces dentro de una misma aplicación), se ha optado por utilizar un catalogo 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.

SENSOR DEVICES			ACTUATOR DEVICES			CONTROLLER DEVICES		
Symbol	Device	Requirement	Symbol	Device	Requirement	Symbol	Device	Requirement
Water Detector	Security		On/Off Light	Lighting		Week Timer	Lighting On/Off
Smoke Detector	Security		Dimmer Light	Lighting		Mobile	Security
Presence Detector	Security / Lighting		Shutter	Motors	Note: Not all the controller devices needed are represented in this table.
Sunset Sensor	Lighting / Motors		Alarm	Security			

Tabla 1. Catálogo de unidades funcionales .

En este sentido, y gracias a la existencia del catálogo, la especificación de una aplicación domótica se realiza mediante la instanciación de unidades funcionales (“*FUnitInstance*”) que vienen definidas en el catálogo. Estas instancias de unidades funcionales deben ser configuradas añadiendo los valores necesarios a sus “*Parámetros*”. Finalmente, mediante enlaces (“*Link*”) se puede indicar la forma en la que las unidades funcionales van a interactuar con el resto del sistema. Al realizar un enlace se indican los servicios que se interconectan. Estos enlaces pueden comportarse como enlaces de tipo canal en el caso de que una de las unidades funcionales involucradas sea pasiva de manera que se modela una conexión hardware o bien como un enlace normal.

Entorno Gráfico

En este documento se presentan entorno integrado para el aprendizaje de las técnicas de diseño de instalaciones domóticas sin profundizar en ninguna plataforma específica, soportando el desarrollo de aplicaciones que pueden ser implementadas de forma real. Este entorno se ha desarrollado como plugin de Eclipse [6], por lo que mantiene una estructura visual muy parecida. Figura 1 muestra la interfaz visual dividida en 3 zonas bien diferenciadas:

- Área de dibujo: zona de trabajo donde se arrastran las figuras desde la paleta para construir un modelo gráfico concreto. También existe la opción de instanciar elementos desde un menú flotante en lugar de utilizar la paleta.
- Paleta gráfica: contiene las primitivas que se pueden arrastrar a la zona de dibujo. Permite mediante “drag and drop” crear elementos, por ejemplo si se pulsa en la paleta sobre la herramienta “StandardFUnitInstance” y acto seguido pulsamos sobre el área de dibujo se creará una nueva instancia de Unidad Funcional.
- Área 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

METODOLOGÍA DE ENSEÑANZA

La nueva metodología de enseñanza aquí propuesta consta de dos roles principales, el del profesor y el del alumno. Si atendemos a las distintas tareas de cada uno de estos roles podemos distinguir las siguientes fases (véase Figura 1)

El profesor impartirá los conceptos teóricos del dominio domótico en diferentes sesiones teóricas. El profesor planteará una serie de problemas a resolver de forma individual por cada uno de los alumnos. Los problemas planteados se corresponden con un caso práctico de implementación de un sistema domótico real (maquetas del laboratorio)

En el laboratorio los alumnos contarán con un ordenador, con la herramienta HABITATION instalada, un catálogo de unidades funcionales reutilizables y una maqueta de una casa domótica.

Los estudiantes deberán resolver el problema práctico con la ayuda de la herramienta e implementar el sistema domótico final correctamente en la maqueta.

Para la resolución del problema propuesto, los alumnos realizarán inicialmente un análisis de requisitos a satisfacer por el sistema domótico. Atenderán a los requisitos de seguridad, confort, comunicación y ahorro energético y decidirán que funcionalidades son necesarias para satisfacer dichos requisitos.

A continuación los alumnos diseñarán el sistema haciendo uso de la herramienta HABITATION. Primero deberán cargar el catálogo de unidades funcionales para poder instanciar y reutilizar dichas unidades en un modelo de aplicación. Modelaran el sistema instanciando unidades funcionales y dando los valores necesarios a sus parámetros.

Por ejemplo, si se desea instanciar una unidad funcional correspondiente a una entrada binaria que actúe como conmutador, se deberá pulsar en la paleta sobre “*StandardFUnitInstance*”, acto seguido se debe pulsar sobre el área de dibujo y automáticamente se creará una nueva instancia de Unidad Funcional. Una vez se tiene la unidad funcional se debe indicar de que unidad funcional del catálogo es instancia, para esto el alumno debe dirigirse al campo “*isInstanceOf*” que aparece en el área de propiedades cuando la unidad funcional está seleccionada. Una vez instanciada la unidad funcional deseada, si los parámetros de esta unidad necesitarán tomar ciertos valores se deberán incorporar pulsando en la paleta sobre “*parameter*” y soltando dentro de la unidad funcional y a continuación sobre el tipo de valor (*Integer*, *Number*, *String*, *Enumeration*) e indicando en el área de propiedades el valor exacto.

Cuando estén todas las unidades funcionales instanciadas y parametrizadas los alumnos podrán enlazar según su criterio dichas unidades, indicando en cada enlace que servicios se utilizan.

En el momento en el que el alumno guarde su modelo de aplicación domótica, la herramienta realizará una validación automática para comprobar que no se han incumplido ninguna restricción impuesta por la herramienta. Por ejemplo, se comprueba que cada unidad funcional instanciada tenga un identificador único dentro de la aplicación.

Llegado a este punto HABITATION realizará una serie de transformaciones internas totalmente transparente para el alumno obteniendo finalmente la generación automática de código del sistema domótico que el alumno ha modelado.

Esté código está listo para ser cargado directamente a los dispositivos domóticos, permitiendo implementar el sistema domótico modelado por el alumno en un caso real. En nuestro caso los alumnos podrán comprobar el correcto funcionamiento del sistema domótico en las maquetas que se les proporciona en el laboratorio.

Durante todo el proceso la herramienta genera una serie de trazas que permite tanto al alumno como al profesor revisar de una manera rápida cada una de los pasos que el alumno a dado hasta llegar a la obtención del código, al igual permite realizar un seguimiento de en que dispositivos se ha implementado cada requisito.

Figura 1. Metodología docente.

REFERENCIAS

- [1] KNX standard disponible en <http://www.knx.org/>
- [2] LonWorks standard disponible en <http://www.lonmark.org/>
- [3] B. Selic, The Pragmatics of Model-Driven Development, IEEE Software, Vol. 20, No. 5, 2003, pp. 19-25.
- [4] M. Mernik, J. Heering, A. M. Sloane, When and How to Develop Domain-Specific Languages, ACM Computing Surveys, Vol. 37, No. 4, 2005, pp. 316-344.
- [5] M. Jimenez, F. Rosique, P. Sánchez, B. Álvarez, A. Iborra, Habitation: A Domain-Specific Language for Home Automation, IEEE Software, Vol. 26, No. 4, 2009, pp. 33-38.
- [6] Eclipse project home. <http://www.eclipse.org>.
- [7] Benito, A. y Cruz, A.; Nuevas claves para la Docencia Universitaria en el Espacio Europeo de Educación Superior. Ed. Narcea, 2005.
- [8] R.J. López, A. Pérez, Innovación docente y Software libre. II Encuentro sobre innovación en docencia universitaria. Universidad de Alcalá. Junio 2007.
- [9] Eclipse Modeling Framework Project. <http://www.eclipse.org/modeling/emf>
- [10] Graphical Modeling Framework Project. <http://www.eclipse.org/gmf>
- [11] E. Biermann, K. Ehrig, C. Ermel, C. Köhler, G. Kuhns, G. Taentzer, Tiger EMF Model Transformation Framework (EMT). <http://tfs.cs.tu-berlin.de/emftrans/papers/userdoc.pdf>.
- [12] Eclipse Consortium, Java Emitter Templates (JET). <http://www.eclipse.org/modeling/m2g/?project=jet>.
- [13] F. Mueller, and A. L. Hosking, Penumbra: An Eclipse plugin for introductory programming, Proceedings of the 2003 OOPSLA Workshop on Eclipse Technology eXchange, 2003, pp. 65-68.
- [14] C. Reis, and R. Cartwright, A friendly face for eclipse, Proceedings of the 2003 OOPSLA Workshop on Eclipse Technology eXchange, 2003, pp. 25-29.
- [15] J. Muñoz, V. Pelechano, and C. Cetina, Implementing a Pervasive Meetings Room: A Model Driven Approach, Proceeding of the 3rd International Workshop on Ubiquitous Computing, 2006, pp. 13-20.
- [16] M. Voelter, and I. Groher, Product line implementation using aspect-oriented and model-driven software development, Proceedings of the 11th International Software Product Line Conference, 2007, pp. 233-242.
- [17] G. Nain, E. Dauber, O. Barais, and J. M. Jézéquel, Using mde to build aschizofrenic middleware for home/building automation, In ServiceWave'08: Networked European Software & Services Initiative (NESSI) Conference, 2008.
- [18] A. Seffah, M. Donyaee, R. B. Kline, and H. K. Padda, Usability Measurement and Metrics: A Consolidated Model, Software Quality Journal, Vol. 14, No. 2, 2006, pp. 159-178.

Introducing Safety Requirements Traceability Support in Model-Driven Development of Robotic Applications

En este artículo se presenta un enfoque basado en la trazabilidad de requisitos de seguridad, desde la fase de requisitos hasta el diseño y desde el diseño hasta la implementación final mediante transformaciones de los distintos modelos implicados. Se demuestra la eficacia y el valor de la combinación de un enfoque basado en modelos junto con el uso de técnicas de trazabilidad en el diseño de sistemas reactivos, en concreto los sistemas de seguridad y como el desarrollo de estos sistemas puede verse beneficiado en gran medida por la combinación de MDE con técnicas relacionadas con el concepto de trazabilidad.

Aunque este artículo no es de aplicación directa al dominio de la domótica, si que presenta el enfoque de trazabilidad para sistemas reactivos (dentro de cuyo ámbito también se encuentra el dominio domótico) que se plantea en mi Tesis. En esta ocasión el caso de estudio utilizado para validar dicho enfoque ha sido un sistema reactivo diferente al domótico, lo que demuestra la validez y generalidad del enfoque propuesto.

El término trazabilidad está definido en el IEEE Standard Glossary of Software Engineering Terminology [11] como: “the degree to which a relationship can be established between two or more products of the development process, especially products having a predecessor successor or master subordinate relationship to one another; for example, the degree to which the requirements and design of a given software component match”. Un entorno de desarrollo MDE puede facilitar en gran medida la trazabilidad, ya que se puede trazar enlaces entre artefactos origen y destino a la vez que se generan otros artefactos software mediante transformaciones de modelos.

Estos enlaces entre artefactos origen y destino pueden ser útiles para la realización del análisis de impacto, la sincronización entre los modelos, depuración de modelos y determinar el origen o el destino de una transformación. Los enlaces serán más fácil de validar si las transformaciones se realizan adecuadamente y si los objetos obtenidos a lo largo del proceso de transformación satisfacen correctamente los requisitos de los interesados.

Los desarrolladores pueden de esta forma validar la integridad y la exactitud de las transformaciones, obtener soporte a la gestión integrada de cambios en los requisitos del sistema y evaluar el impacto de los cambios en los distintos niveles de abstracción (requisitos, diseño arquitectónico, diseño detallado y, finalmente, código ejecutable).

Para implantar un enfoque que integre seguridad, MDE y trazabilidad es necesario contar con una serie de recursos tanto desde el punto de vista conceptual como de desarrollo (técnicas, herramientas, etc.). Básicamente se presentan como imprescindibles:

1. Un marco integrado de gestión de requisitos de seguridad.
2. Un catálogo de soluciones arquitectónicas para dichos requisitos.

3. Un marco para la generación automática de código a distintas plataformas de implementación.
4. Soporte para la trazabilidad de los distintos artefactos involucrados en el desarrollo.
5. Herramientas que permitan la manipulación completa y automatizada de toda la información involucrada en el proceso.

Este artículo presenta un entorno integrado que proporciona estos recursos y demuestra la viabilidad del enfoque con la experiencia adquirida en el desarrollo de un sistema robótico industrial con requisitos de seguridad.

FRAMEWORK DE GESTION DE REQUISITOS DE SEGURIDAD

Como se indicó anteriormente, la gestión de requisitos safety desde una perspectiva MDE requiere el diseño de un metamodelo para representarlos. Cada sistema que se construya tendrá definidos un conjunto de requisitos de seguridad, posiblemente reutilizados de experiencias similares o incluso de catálogos de terceras partes. Estos requisitos se corresponden con el “*Safety Requirement Model*” que es conforme, siguiendo la nomenclatura de MDE, con su correspondiente metamodelo.

El proceso de desarrollo se aplica por tanto como se muestra en la Figura 1.

Figura 1. Marco general MDE de desarrollo de aplicaciones basadas en componentes, completado con la gestión de trazabilidad de requisitos safety.

El usuario comienza modelando los requisitos safety o reutilizando aquellos que previamente hubiera definido. Posteriormente, debe desarrollar una solución arquitectónica (un modelo V3CM) que satisfaga los requisitos de seguridad utilizando para ello patrones conocidos [4] para el diseño arquitectónico de sistemas safety. Las correspondencias de trazabilidad entre requisitos safety y los elementos del diseño arquitectónico que los satisfacen deben ir anotándose en el modelo llamado

“*Traceability Model #1*”, conforme a un metamodelo de trazabilidad específico para este framework.

Es importante destacar que, tal y como se plantea en la metodología, ésta permite (1) aprovechar catálogos de requisitos y soluciones, y (2) trazar las decisiones arquitectónicas y de implementación de manera casi automática.

Trazabilidad de requisitos de seguridad

La trazabilidad de artefactos software permite conocer con mucho mayor detalle la conformidad del sistema desarrollado con sus requisitos, así como las implicaciones que puede tener un cambio. Para poder guardar y posteriormente procesar los vínculos de trazabilidad definidos es necesario disponer de un repositorio de trazas (links) entre los distintos artefactos software generados.

La única forma de poder integrar la trazabilidad en una propuesta MDE es definiendo un metamodelo (véase Figura 2) que permita modelar las relaciones relevantes entre los artefactos origen y destino implicados en las distintas transformaciones realizadas.

Figura 2. Metamodelo de trazabilidad.

Lo realmente interesante del enfoque de trazabilidad utilizado es que una vez se ha generado el modelo de trazabilidad es posible obtener un informe detallado de los elementos trazados utilizando la herramienta SRTT (*Safety-Requirements-Trace-Tool*) desarrollada (véase Figura 3). La herramienta, desarrollada utilizando JET, genera el informe de trazabilidad en formato HTML aplicando una transformación M2T a los modelos de trazabilidad existentes. Este informe permite a los diseñadores estudiar cómo un requisito se ha sido tenido en cuenta en el proceso de generación automática de código y proporciona información de la solución adoptada, que en algunos casos será un patrón de diseño.

Trace Link		Source Link		Target Link		
Name	Description	Metamodel	Model Element	Metamodel	Model Element	Composite?
H13 - watchdog	This link refers to a Component, three ports and two interface					✓
H13 to Component	Add additional emergency stop mechanisms. Add external motion sensors (6.4 ANSI). Watchdog implementation.	safety	safety:hasHAZARD - origins => Error in the primary positioning system controller. Communications failure. Power failure - reactionMeasures => Emergency alarm. Power robot off. Emergency stop. - description => The primary positioning system does not stop - id => H13	v3cm	SimpleComponent - name => Watchdog	Child of: H13 - watchdog
H13 to Port	The Watchdog Unit Component will receive commands from the MMI through this port	safety	safety:hasHAZARD - origins => Error in the primary positioning system controller. Communications failure. Power failure - reactionMeasures => Emergency alarm. Power robot off. Emergency stop. - description => The primary positioning system does not stop - id => H13	v3cm	component:port - name => P_Wdog_Cmd	Child of: H13 - watchdog
H13 to Port	The Watchdog Unit Component will receive the signal for implementing the watchdog pattern.	safety	safety:hasHAZARD - origins => Error in the primary positioning system controller. Communications failure. Power failure - reactionMeasures => Emergency alarm. Power robot off. Emergency stop. - description => The primary positioning system does not stop - id => H13	v3cm	component:port - name => P_Wdog_Wdog	Child of: H13 - watchdog

Figura 3. Fragmento del informe de trazabilidad generado con la herramienta RTT.

La Figura 3 muestra una captura de pantalla del uso de la herramienta SRTT, donde se pueden apreciar los elementos almacenados en el “Traceability Model #1”. Para cada una de las trazas se detalla su nombre y descripción, así como la información relativa al origen (*source*) y el destino (*target*) de los elementos trazados. De esta manera, el informe de trazabilidad permite al usuario distinguir los requisitos que han sido considerados, los elementos arquitectónicos que dan soporte a cada uno de ellos, los elementos del ámbito de la orientación a objetos que están asociados, y por último, las porciones del código en que se han implementado.

La aplicación de este enfoque de trazabilidad permite obtener los siguientes beneficios:

- Apoyo a la validación de los requisitos de seguridad: La validación de los requisitos puede realizarse a través del informe de trazabilidad descrito anteriormente. En el informe se puede comprobar que todos los requisitos de seguridad considerados están representados en algún elemento arquitectónico del modelo V3CM. De cara a comprobar que un requisito safety ha sido considerado en su totalidad, es posible mostrar únicamente aquellos elementos arquitectónicos que tienen que ver con dicho requisito.
- Comprobar la compatibilidad entre el modelo arquitectónico y el modelo de requisitos de seguridad: El informe de trazabilidad ofrece al analista las correspondencias explícitas entre los requisitos safety y cada uno de los elementos de la solución arquitectónica diseñada. Para poder realizar esta verificación, el analista debe tener perfectamente catalogados los patrones utilizados para satisfacer los requisitos safety. Estos patrones son soluciones bien conocidas en el ámbito de los sistemas safety. La verificación de los requisitos necesita tanto del informe de trazabilidad como del detalle de los patrones utilizados en las reglas de transformación definidas. Sin embargo, la verificación de la corrección no es inmediata. Estas características pueden verificarse más fácilmente a partir de la información de traza proporcionada por la herramienta SRTT.

- Permite establecer el impacto del cambio de un requisito de seguridad: es habitual que los usuarios del sistema sugieran cambios en los requisitos a lo largo del proceso de desarrollo. El informe de trazabilidad puede ayudar a los analistas a evaluar el impacto de estos cambios antes de tener que aplicarlos. El primer dato de utilidad para el analista es saber cuántos elementos de cada nivel deben modificarse, teniendo en cuenta que las modificaciones se propagan hacia los modelos de niveles inferiores. Por otro lado, el analista puede igualmente verificar la implicación que tiene la modificación de un requisito sobre el resto de requisitos ya implantados. En este sentido, es importante conocer los compromisos entre los atributos de calidad definidos en el sistema. La herramienta SRTT puede ayudar al ingeniero en esta toma de decisiones puesto que puede ver más claramente las relaciones de dependencia entre los requisitos originales y las decisiones arquitectónicas empleadas para cumplirlos. Por ejemplo, la incorporación del componente WathdogUnit para dar solución a la amenaza H13 puede repercutir negativamente en el rendimiento puesto que este requisito depende fundamentalmente de la carga de comunicaciones existente. Esta toma de decisiones puede verse enriquecida con distintas propuestas que ligán soluciones arquitectónicas con atributos de calidad [25]. Estas propuestas pueden usarse para crear definiciones preliminares de la arquitectura software, analizar decisiones arquitectónicas en ausencia de código y analizar diversas arquitecturas candidatas para el desarrollo de un sistema. El propósito principal es descubrir las consecuencias de las decisiones arquitectónicas a la luz de los atributos de calidad. El estudio de estos atributos debe hacerse teniendo en cuenta la dependencia con el resto de atributos, puesto que no son independientes sino que interaccionan entre sí a través de las relaciones estructurales impuestas por la arquitectura.

REFERENCIAS

- [1] A. Iborra, D. Alonso, F. Ortiz, J. Franco, P. Sánchez, and B. Álvarez, "Design of service robots," *IEEE Robot. Automat. Mag., Special Issue on Software Engineering for Robotics*, vol. 16, no. 1, pp. 24–33, Mar. 2009.
- [2] C. Fernández, A. Iborra, B. Álvarez, J. Pastor, P. Sánchez, J. Fernández, and N. Ortega, "Ship shape in europe: cooperative robots in the ship repair industry," *IEEE J. Robot. Automat.*, vol. 12, no. 3, pp. 65–77, Sep. 2005.
- [3] American National Standard Institute, "ANSI/RIA R15.06: American National Standard for Industrial Robots and Robot Systems – Safety Requirements," 1999.
- [4] B. Douglass, *Real-Time Design Patterns: Robust Scalable Architecture for Real-Time Systems*, ser. Object Technology. A-W Prof., 2002, iISBN: 0-201-69956-7.
- [5] D. Alonso, P. Sánchez, F. Ortiz, J. Pastor, B. Álvarez, and A. Iborra, Service robot applications. In-Teh, 2008, ch. Experiences developing safe and fault-tolerant tele-operated service robots. A case study in shipyards, pp. 159–182.
- [6] D. Alonso, P. Sánchez, B. Álvarez, and J. Pastor, "A systematic approach to developing safe tele-operated robots," in *11th International Conference on Reliable Software Technologies*, ser. LNCS, vol. 4006, Jun. 2006, pp. 119–130.
- [7] B. Selic, "The pragmatics of model-driven development," *IEEE Trans. Software Eng.*, vol. 20, no. 5, pp. 19–25, 2003.
- [8] C. Atkinson and T. Kühne, "Model-driven development: a metamodeling foundation," *IEEE Softw.*, vol. 20, no. 5, pp. 14–18, 2003.

- [9] T. Mens and P. van Gorp, "A taxonomy of model transformation," *Electronic Notes in Theoretical Computer Science*, vol. 152, pp. 125–142, 2006.
- [10] D. Alonso, "Software development for service robots: a modeloriented and component-based approach," Ph.D. dissertation, Information Technologies and Communications Department, Universidad Politécnica de Cartagena (Spain), 2008.
- [11] IEEE 610.12-1990 Standard Glossary of Software Engineering Terminology, IEEE.
- [12] A. Clifton and I. Ericson, *Hazard Analysis Techniques for System Safety*. Wiley, 2005.
- [13] European Committee for Electrotechnical Standardization, "EN 61508: Functional Safety of Electrical/Electronic/Programmable Electronic Safety-Related Systems," 2003.
- [14] B. Douglass, *Doing hard time: developing real-time systems with UML, objects, frameworks and patterns*, ser. Object Technology. Addison-Wesley Longman Publishing Co., Inc., 1999, ISBN: 0-201-49837-5.
- [15] D. Schmidt, M. Stal, H. Rohnert, and F. Buschmann, *Pattern-Oriented Software Architecture, Vol. 2: Patterns for Concurrent and Networked Objects*. Wiley, 2000.
- [16] C. Szyperski, *Component Software: Beyond Object-Oriented Programming*, 2nd ed. A-W, 2002.
- [17] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design patterns: elements of reusable object-oriented software*. A-W Prof., Jan. 1995.
- [18] D. Steinberg, F. Budinsky, M. Paternostro, and E. Merks, *EMF: Eclipse Modeling Framework*, 2nd ed., A.-W. Prof., Ed., 2008.
- [19] F. Jouault, F. Allilaire, J. Bézivin, and I. Kurtev, "Atl: A model transformation tool," *Science of Computer Programming*, vol. 72, no. 1-2, pp. 31–39, 2008.
- [20] A. Egyed and P. Grünbacher, "Automating requirements traceability: beyond the record & replay paradigm," in *Proc. of the 17th IEEE International Conference on Automated Software Engineering*. IEEE, 2002.
- [21] V. Vicente-Chicote, B. Moros, and A. Toval, "Remm-studio: an integrated model-driven environment for requirements specification, validation and formatting," *Journal of Object Technology*, vol. 6, no. 9, pp. 437–454, Oct. 2007.
- [22] A. Toval, B. Moros, J. Nicolás, and J. Lasheras, "Eight key issues for an effective reuse-based requirements process," *International Journal of Computer Systems Science and Engineering*, vol. 23, no. 6, pp. 1–20, Nov. 2008.
- [23] O. Gotel, "Contribution structures for requirements traceability," Ph.D. dissertation, Department of Computing, Imperial College, London (England), 1995.
- [24] D. Kolovos, R. Paige, and F. Polack, "On-demand merging of traceability links with models," in *Proc. of the 2nd EC-MDA Workshop on Traceability*, 2006.
- [25] R. Kazman, L. Bass, and M. Klein, "The essential components of software architecture design and analysis," *Journal of Systems and Software*, vol. 79, no. 8, pp. 1207–1216, Aug. 2006. *IEEE TRANSACTIONS ON COMPUTERS*, JUNE 2009 13

A framework for developing home automation systems: from requirements to code

En este artículo se presenta un framework para el desarrollo de sistemas domóticos siguiendo un enfoque de arquitectura dirigida por modelos (MDA – Model Driven Architecture). Este framework da soporte completo a toda la metodología de desarrollo propuesta en esta Tesis, desde la captura de requisitos hasta la generación del código de la aplicación domótica final. Este artículo es por tanto el más representativo y en el que se agrupa la mayor parte del conocimiento generado a lo largo del trabajo de Tesis, por ello se hace referencia a ideas que ya se han ido presentando de forma separada en otros artículos, como es el caso del DSL HABITATION, la propuesta de trazabilidad o la herramienta RTT (Requirement Trace Tool).

En este documento se presenta el framework junto con las distintas herramientas creadas para cubrir las necesidades de cada fase del ciclo de vida. Estas herramientas incluyen (1) un DSL para la creación de modelos de requisitos domóticos, (2) un modelo intermedio de componentes para reducir la complejidad y facilitar la confluencia con otros dominios de sistemas reactivos, (3) modelos para las plataformas específicas de ejecución y (4) un conjunto de transformaciones para refinar los modelos de requisitos, pasando por los diferentes niveles de abstracción, hasta obtener código ejecutable. Y todo esto extendido con mecanismos de trazabilidad de todos los modelos involucrados en el proceso. Para el desarrollo de este framework se presentan por tanto como imprescindibles:

1. Un marco integrado de gestión de requisitos para el dominio domótico,
2. un catálogo de dichos requisitos expresados mediante el DSL,
3. un modelo de componentes que permita captar tanto aspectos estructurales como de comportamiento,
4. un marco para la generación automática de código a distintas plataformas de implementación,
5. soporte para la trazabilidad de los distintos artefactos involucrados en el desarrollo y, finalmente,
6. herramientas que permitan la manipulación completa y automatizada de toda la información involucrada en el proceso.

En resumen, este artículo contribuye al estado de la técnica con las siguientes características:

- Un marco que integra un conjunto de herramientas para la definición de las aplicaciones domóticas a diferentes niveles de abstracción.
- Un conjunto de transformaciones de modelos [7] que permite a los desarrolladores obtener el código final ejecutable.

- Soporte de trazabilidad [8] para mejorar la calidad tanto del proceso como de los modelos obtenidos.
- Un estudio que demuestra el éxito de la reutilización de modelos en MDD por medio de requisitos genéricos. Investigamos los factores que contribuyen a este éxito.

Este artículo presenta un entorno integrado que proporciona todos estos recursos y demuestra la viabilidad del enfoque con la experiencia adquirida en su aplicación a un caso de estudio que recoge un conjunto representativo de requisitos en el campo de la domótica.

El framework para sistemas domóticos

La Figura 1 muestra el marco completo de desarrollo de sistemas domóticos siguiendo un enfoque MDA. En ella pueden observarse los distintos pasos que hay que dar a lo largo del proceso de desarrollo utilizando este framework. También se puede identificar los distintos niveles de abstracción, que corresponden a los definidos por el enfoque MDA. Los distintos artefactos que podemos encontrar en estos niveles se corresponden con: (1) requisitos reutilizables, (2) modelo de la aplicación domótica en notación del DSL HABITATION(Jimenez et al. 2009b), (3) modelo de componentes, (4) modelo específico de la plataforma. Cada modelo construido es conforme a su propio metamodelo. Cada requisito tiene asociado un fragmento de DSL, de esta manera, quedan catalogadas las soluciones parciales a cada requisito domótico. Al construir una nueva aplicación, el usuario debe inspeccionar dicho catálogo de requisitos identificando los requisitos que va a reutilizar del mismo. Al hacerlo, reutiliza las soluciones adoptadas con el DSL en el desarrollo de aplicaciones anteriores. Entre el modelo de requisitos y el modelo de la aplicación DSL las transformaciones se realizan de forma manual. Desde el modelo DSL hasta el modelo específico de la plataforma (por el momento sólo para la plataforma KNX/EIB), las transformaciones modelo a modelo (M2M) se realizan de manera automática. En cada transformación (tanto manual como automática), ya sea entre modelos de distintos niveles o bien del mismo nivel de abstracción, se registra un modelo de trazabilidad, también conforme a un metamodelo que se describe a continuación, indicando la evolución de cada uno de los artefactos software involucrados desde origen hasta su destino. Los modelos de trazabilidad son posteriormente procesados por la herramienta RTT (Requirement Tracer Tool) que proporciona al usuario una serie de informes de trazabilidad útiles para distintas situaciones. Finalmente el framework permite obtener, mediante una transformación modelo a texto (M2T), el código ejecutable de la aplicación domótica.

Figura 1: Resumen del framework.

Una de las principales aportaciones de este framework es la existencia de un catálogo 'genérico' de requisitos domóticos reutilizables, que pueden ser utilizados para la creación de instancias posteriores del sistema, contribuyendo significativamente a la reutilización simplemente seleccionando un subconjunto de estos requisitos para el desarrollo del nuevo sistema. Como se ha dicho anteriormente, cada requisito tiene asociado un fragmento de modelo en notación DSL. Un fragmento forma parte de un modelo completo en el sentido de que lleva a cabo parte de la funcionalidad deseada. Es posible que un fragmento sea sintáctica o semánticamente incompleto. Por lo tanto, las reglas de integridad deben estar desactivadas temporalmente con el fin de facilitar la integración de estos fragmentos en un modelo de aplicación.

A continuación se verán cada uno de las partes involucradas del framework en más detalle.

- **Gestión de Requisitos:** con el fin de integrar la gestión de requisitos en el proceso, ha sido necesario adoptar un metamodelo. La Figura 2 muestra el metamodelo definido para la representación de los requisitos domóticos. Este metamodelo (llamado "Requirement Metamodel") está integrado en el entorno de desarrollo presentado en la sección anterior (Figura 2) y tiene el grado de complejidad y versatilidad suficiente para poder adoptar un enfoque de reutilización de requisitos que sea válido para otros dominios, como se demostró en el (Sanchez et al. 2011), y, al mismo tiempo, sencillo en su estructura.

Figura 2: Metamodelo de requisitos.

La clase raíz la constituye la clase *Catalogue* donde se integran un conjunto de requisitos (*Requirement*). Cada uno de estos requisitos incluye un nombre, una descripción y el procedimiento de validación que debería seguir el usuario para, una vez desplegado el sistema final, comprobar que el sistema integra la funcionalidad prevista. Por ejemplo, ante el requisito “Debe contarse con detección de humos”, el procedimiento de validación establecería “Provocar una situación simulada con humo y verificar que saltan las alarmas correspondientes”. Los requisitos no están aislados sino que se relacionan con el resto. Para ello la clase *Relationship* permite ligar un requisito (source relationship) con uno o más requisitos destino (target relationship). El tipo correspondiente a la relación entre requisitos queda catalogado con la clase *TypeOfRelationship*.

- **Lenguaje específico del Dominio:** En el framework se incluye el DSL HABITATION (para más detalle véase (Jimenez et al. 2009b)) para modelar la aplicación de una manera fácil e intuitiva y además de forma gráfico. Como se ha comentado anteriormente, el modelo de requisitos se representa con este DSL. Como cada requisito está representado por un fragmento de modelo DSL, puede darse el caso que en una aplicación varios requisitos compartan las mismas unidades funcionales y producirse por tanto una superposición entre estas unidades funcionales. Por esta razón es necesaria realizar un refinamiento manual del modelo de aplicación generado a partir del modelo de requisitos. En la Figura 3 se puede apreciar como los distintos requisitos seleccionados se han ensamblado dando lugar a un modelo de aplicación DSL.
- **Transformaciones de modelos:** las transformaciones entre los modelos DSL y los modelos componentes se han realizado usando un enfoque basado en gramáticas de grafos [24] (en particular, el plugin EMT [25] para el entorno de Eclipse). El hecho de que los modelos suelen estar representados de forma gráfica, hace más atractivo el uso de este tipo de enfoque para las transformaciones, dado que son más fáciles de entender y seguir. La transformación del modelo DSL al modelo de componentes se realiza mediante una transformación que contiene varias reglas. Cada regla (véase la figura 4) se separa en lado izquierdo (LHS), donde se indican las pre-condiciones para la transformación en forma de grafo patrón, y un lado derecho (RHS), mostrando el grafo resultado de la transformación. Tanto el LHS y RHS son los gráficos, los cuales suelen contener elementos de la entrada y la salida de los modelos. La inclusión de elementos del modelo de producción en el LHS se asegura de que las transformaciones que se ejecutan en la secuencia correcta. Una regla

también puede tener una condición de no aplicación (NAC), que no deben cumplirse para su aplicación.

Figura 3: Aplicación resultante en el modelo DSL tras seleccionar 3 requisitos.

Figura 4. Ejemplo de regla de transformación.

De esta manera se obtiene un modelo de componentes a partir del DSL. A continuación se realiza otra transformación de este modelo de componentes para obtener un modelo específico de la plataforma de.

- **Trazabilidad:** dada la amplia utilización de transformaciones y la rapidez con la que se modifican los distintos artefactos software involucrados, es necesario contar con mecanismos de control que permitan rastrear dicho artefacto a lo largo de todo el proceso de desarrollo. En este sentido, la trazabilidad de artefactos software permite conocer con mucho mayor detalle la adecuación del sistema desarrollado así como las implicaciones que puede tener un cambio. Podemos resumir estas capacidades en los siguientes elementos:

1. Ser capaz de validar si los distintos requisitos han sido considerados y si la implementación obtenida es compatible con ellos.
2. Verificar la necesidad de cada requisito y de qué manera ha sido satisfecho.
3. Conocer el impacto que puede tener modificar un requisito.

El framework expuesto en este documento, integra su propio mecanismo de trazabilidad. Para ello se ha definiendo un metamodelo, el “*Traceability metamodel*” que se detalla en (Sanchez et al. 2011), con él se pueden definir modelos de trazabilidad donde reflejar las distintas transformaciones que han sufrido los artefactos a lo largo de todo el proceso. Los modelos de trazabilidad se crean durante el proceso de transformación de modelos. El primer modelo de trazabilidad (entre requisitos domóticos y el modelo correspondiente al DSL) es creado manualmente por el usuario. Desde el DSL hasta el código final generado, todas las trazas se obtienen automáticamente como parte del proceso de transformación de modelo origen a destino. Para disponer de esta facilidad de traza deben extenderse las reglas de transformación de grafos definidas con una parte dedicada a anotar la traza en el metamodelo definido.

Por último, una vez se ha generado el correspondiente modelo de trazabilidad es posible obtener un informe detallado de los elementos trazados utilizando la herramienta RTT desarrollada (Requirements-Trace-Tool) (para más detalle véase (Sanchez et al. 2011)). Este informe permite a los diseñadores de sistemas domóticos analizar cómo un requisito ha sido tenido en cuenta en el proceso de generación automática de código e información de la solución adoptada. La Figura 5 muestra una captura de pantalla del uso de la herramienta RTT, donde se pueden apreciar los elementos almacenados en el “Traceability Model #1”. Para cada una de las trazas se detalla su nombre y descripción, así como la información relativa al origen (source) y el destino (target) de los elementos trazados.

Trace Link		Source Link		Target Link		
Name	Description	Metamodel	Model Element	Metamodel	Model Element	Composite?
Requisito Luz	Transformación de requisitos de luz a los fragmentos correspondientes de la aplicación domótica (aplicación DSL)					✓
Req3 to DSL	Transformación del requisito 3 al DSL	Requirement	<u>Requirement:hasRequirements</u> - name => Encendido/Apagado automático luz - description => Encendido/Apagado de un punto de luz de forma automática por detección de presencia - validationProcedure => Entrar en una habitación y comprobar que se enciende automáticamente la luz - id => REQ3	MetaDomo	<u>MetaDomo:hasFuncInstance</u> - FInstanceName => PIR3-2 - FInstanceRef =>	Child of: Requisito Luz
				MetaDomo	<u>MetaDomo:hasFuncLink</u> - FLinkName => C3 - canal => true	Child of: Requisito Luz
				MetaDomo	<u>MetaDomo:hasFuncInstance</u> - FInstanceName => SW1-1 - FInstanceRef =>	Child of: Requisito Luz

Figura 5: Informe de trazabilidad.

- **Generación de código:** Finalmente el proceso termina con la generación de código y la implementación real del sistema domótico. En esta fase se ha apostado por el aprovechamiento de las herramientas disponibles en cada una de las tecnologías para la obtención del código en lugar de construir un nuevo conjunto de herramientas a partir de cero. En el caso de la tecnología KNX, en la actualidad el punto de partida para un desarrollador tradicional sería la creación del proyecto

mediante la iteración manual con la herramienta ETS. Con la nueva metodología que se propone en este trabajo de Tesis, esa iteración desaparece y la herramienta se convierte en un soporte para la ejecución del código obtenido.

El plugin ITTool de la herramienta ETS proporciona un entorno de desarrollo para escribir el código de programación de los sistemas domóticos. Este código se escribe en el lenguaje VBscript y se organiza en una serie de ficheros llamados macros. Estas macros son ejecutadas por la herramienta ETS obteniendo así la configuración final que se implementa en cada uno de los dispositivos. La metodología propuesta para esta fase se resume entonces en la transformación con JET modelo-a-texto para obtener a partir del modelo específico de la plataforma las macros automáticamente (véase Figura 6). Estas macros se ejecutarán en ETS obteniendo así la configuración del proyecto domótico.

Figura 6: Esquema de la generación de código para la plataforma KNX

REFERENCIAS

- [1] Chana, M., Campoa, E., Estevea, D., Fourniols J.Y., 2009. Smart homes: current features and future perspectives. Elsevier Maturitas 64(2), 90-97.
- [2] Miori, V., Tarrini, L., Manca, M., Tolomei, G., 2006. An open standard solution for domotic interoperability. IEEE Transactions on Consumer Electronics 52(1), 97-103.
- [3] Selic, B, 2003. The pragmatics of model-driven development, IEEE Software20(5), 19-25.

- [4] Mernik, M., Heering, J.A., Sloane, M., 2005. When and how to develop domain-specific languages, *ACM Computing Surveys* 37(4), 316-344.
- [5] Kelly, S., Tolvanen, J.P., 2008. Domain-specific modeling: enabling full code generation. John Wiley & Sons.
- [6] Hermans, F., Pinzger, M., van Deursen A., 2009. Domain-specific languages in practice: A User Study on the Success Factors. *LNCS5795*, 423-437.
- [7] Mens, T., van Gorp, P., 2006. A taxonomy of model transformation. *Electronic Notes in Theoretical Computer Science* 152, 125-142.
- [8] Ramesh, B., Jarke, M., 2001. Toward reference models for requirements traceability. *IEEE Transactions on Software Engineering* 27(1), 58-93.
- [9] Muñoz, J., Pelechano, V., Cetina, C., 2006. Implementing a pervasive meetings room: a model driven approach. In: *Proceeding of the 3rd International Workshop on Ubiquitous Computing*.
- [10] Voelter, M., Groher, I., 2007. Product line implementation using aspect-oriented and model-driven software development. In: *Proceedings of the 11th International Software Product Line Conference*.
- [11] Nain, G., Dauber, E., Barais, O., Jézéquel, J.M., 2008. Using mde to build aschizofrenic middleware for home/building automation. In: *ServiceWave'08: Networked European Software & Services Initiative (NESSI) Conference*.
- [12] Jammes, F., Mensch, A., Smit, H., 2005. Service-oriented device communications using the devices profile for web services. In: *Proceedings of the 3rd international workshop on Middleware for pervasive and ad-hoc computing*.
- [13] vonKnethen, A., Paech, B., Kiedaisch, F., Houdek, F., 2002. Systematic requirements recycling through abstraction and traceability. In: *10th Intl. Requirements Engineering Conf (RE'02)*, pp. 273-281.
- [14] Clements, P., Northrop, L., 2002. *Software product lines: practices and patterns*. Addison-Wesley. Boston.
- [15] Lam, W., 1998. A case-study of requirements reuse through product families. *Annals of Software Engineering* 5, 253-277.
- [16] Bühne, S., Halmans, G., Lauenroth, K., Pohl, K., 2006. Scenario-based application requirements engineering. In: *Software Product Lines - Research Issues in Engineering and Management*, Springer, Heidelberg, pp. 161-194.
- [17] Jacob, L., Reed, K., 2000. Requirements classification and reuse: crossing domain boundaries. In: *6th International Conference on Software Reuse: Advances in Software Reusability table of contents*, LNCS, 1844; 2000, 190-210.
- [18] Lam, W.J., McDermid, J.A., Vickers, A.J., 1997. Ten steps towards systematic requirements reuse. *Journal of Requirements Engineering* 2, 102-113.

- [19] DSM Forum. 2010. <http://www.dsmforum.org>.
- [20] Kärnä, J., Tolvanen, J.P., Kelly, S., 2010. Evaluating the use of domain-specific modeling in practice. <http://www.dsmforum.org/events/DSM09/Papers/Karna.pdf>.
- [21] Jimenez, M., 2009. Development of HA applications using a model driven approach. Ph.D. Thesis. Technical University of Cartagena, Spain, (March). URL <http://hdl.handle.net/10317/846>.
- [22] Vicente-Chicote, C., Moro, B., Toval, A., 2007. Remm-studio: an integrated model-driven environment for requirements specification, validation and formatting. *Journal of Object Technology* 6(9), 437-454.
- [23] Molina, F., Toval, A., 2009. Integrating usability requirements that can be evaluated in design time into Model Driven Engineering of Web Information Systems. *Advances in Engineering Software* 40, 1306-1317.
- [24] Rozenberg, G., 1997. Handbook of graph grammars and computing by graph transformation. World Scientific, Netherlands.
- [25] Biermann, E., Ehrig, K., Ermel, C., Köhler, C., Kuhns, G., Taentzer, G., 2006. Tiger EMF model transformation framework (EMT). <http://tfs.cs.tu-berlin.de/emftrans/papers/userdoc.pdf>.
- [26] Eclipse Consortium, Java Emitter Templates (JET). <http://www.eclipse.org/modeling/m2t/?project=jet>.
- [27] ITTools, IT Gesellschaft für Informationstechnik mbH, <http://www.it-gmbh.de/en/products/ittools.htm>.
- [28] Valderas, P., Pelechano, V., 2009. Introducing requirements traceability support in model-driven development of web applications. *Information and Software Technology* 51(4), 749-768.
- [29] Behrens, T., 2007. Never Without a trace: Practical advice on implementing traceability. <http://www.ibm.com/developerworks/rational/library/feb07/behrens>.
- [30] Olsen, G.K., Oldevik, J., 2007. Scenarios of traceability in model to text transformations. In: 3rd European Conference on Model-Driven Architecture Foundations and Applications, Israel.
- [31] Kolovos, D., Paige, R., Polack, F., 2006. On-demand merging of traceability links with models. In: Proceedings of the 2nd EC-MDA Workshop on Traceability, Bilbao, Spain.
- [32] Fabro D., Bezivin J., Valduriez P., 2006. Weaving models with the eclipse amwplugin. In: Eclipse Modeling Symposium, Eclipse Summit Europe Esslingen, Germany.
- [33] Aizenbud-Reshef N., Nolan, B.T., Rubin, J., Shaham-Gafni, Y., 2006. Model traceability. *IBM Systems Journal* 45(3), 515-26.

- [34] Dick, J., 2002. Rich traceability. In: Proceedings of automated software engineering, California, USA.
- [35] Walderhaug, S., Johansen, U., Stav, E., Agedal, J., 2008. Towards a generic solution for traceability in MDD. In: Proceedings of the third ECMDA traceability workshop (ECMDA-TW).
- [36] Melby, S., 2007. Traceability in Model Driven Engineering. Master Thesis. University Of Oslo, Norway. <http://urn.nb.no/URN:NBN:no-18721>.
- [37] Oldevik, J., Neple, T., 2006. Traceability in model to text transformations. In: 2nd European Conference on Model-Driven Architecture Foundations and Applications, Spain.

6. Conclusiones / Conclusions

No existe una metodología universal para hacer frente a cualquier proyecto de desarrollo de software, por lo que cualquier enfoque debe ser adaptado al contexto del proyecto (recursos técnicos y humanos, tiempo de desarrollo, tipo de sistema, etc.). Históricamente, los métodos tradicionales de desarrollo han intentado abordar la mayor cantidad de situaciones de contexto del proyecto, exigiendo un esfuerzo considerable para ser adaptadas, sobre todo en proyectos pequeños y con requisitos muy cambiantes.

En esta Tesis se ha definido con éxito una propuesta de desarrollo provisto del correspondiente soporte tecnológico para el desarrollo integral de sistemas domóticos utilizando un enfoque dirigido por modelos junto con otras técnicas tradicionales de la ingeniería del software (Ingeniería de Requisitos, trazabilidad, lenguajes específicos de dominio), poniendo de manifiesto la idoneidad de la utilización de las nuevas técnicas (como es el caso de MDE) para solventar problemas reales de implementación en ámbitos puramente ingenieriles donde tradicionalmente no se aplicaba la Ingeniería del Software. Además, se resaltan las ventajas obtenidas al aplicar de forma sinérgica técnicas novedosas con las tradicionales, siendo los resultados realmente prometedores.

Las herramientas y técnicas proporcionadas son sencillas de usar y requieren un tiempo de aprendizaje reducido, lo que repercute en una reducción en los costes de implantación. El desarrollo de este tipo de sistemas ha sido tradicionalmente un trabajo realizado por desarrolladores con alto grado de experiencia en la programación y en la plataforma de ejecución. El enfoque propuesto en esta Tesis permite a los desarrolladores menos experimentados implementar un sistema domótico de una forma fácil e intuitiva, puesto que cuentan con un lenguaje gráfico específico del dominio y las transformaciones se realizan de forma automática y transparente al desarrollador para obtener finalmente el código del sistema.

Otra característica de la propuesta es la apuesta por la reutilización para obtener un resultado más eficiente y productivo. Esta reutilización se aplica a: (1) los distintos artefactos software involucrados a lo largo de todo el proceso (requisitos, unidades funcionales y componentes), (2) a la reutilización de diseños y (3) a las herramientas e infraestructura de ejecución que ya proporcionan los fabricantes de dispositivos. Para ello se ha proporcionado el mecanismo para definir los distintos catálogos de requisitos, de unidades funcionales y de componentes reutilizables, se ha asociado a cada requisito un fragmento de modelo DSL y se han aprovechado los recursos existentes en cada plataforma de implementación. Por ejemplo para KNX se utiliza la herramienta ETS como soporte para la ejecución de las macros obtenidas.

Los catálogos pueden ampliarse sin necesidad de modificar los mecanismos ofrecidos, de manera que siempre estén actualizados y los desarrolladores puedan reutilizar soluciones parciales de otras aplicaciones (por ejemplo a través de los fragmentos DSL asociados a los requisitos) como base de un nuevo desarrollo. Esto ayuda a mejorar la calidad de los modelos y evita tener que repetir el modelado de funciones ya utilizadas

con anterioridad. Esta sinergia entre la utilización de requisitos y DSL es uno de los elementos clave para la reutilización en esta propuesta.

Aunque en muchos casos la creación de un DSL puede suponer una tarea laboriosa, en este dominio los beneficios obtenidos compensan el esfuerzo. En un enfoque de desarrollo dirigido por modelos la utilización de lenguajes específicos del dominio es la opción más adecuada para este tipo de sistemas, donde los conceptos propios están bien delimitados. De esta manera, se permite a los desarrolladores con cierta experiencia en el dominio domótico crear la descripción de los sistemas utilizando un lenguaje específico que será transformado automáticamente en código ejecutable.

En cuanto a las transformaciones, las gramáticas de grafos, utilizadas como primera aproximación al inicio de esta Tesis, son una buena opción dado el carácter gráfico tanto de la transformación como de los propios modelos. No obstante, para trabajos con modelos con un cierto volumen de grafos, la ventaja visual desaparece y se convierte en un inconveniente, siendo casi ilegible el resultado obtenido. Por otro lado, la necesidad de trabajar con una herramienta integrada dentro de Eclipse hace que se contemplen como otros lenguajes declarativos / imperativos, optando finalmente por ATL.

Dado que el dominio domótico supone un subdominio concreto de sistemas reactivos, las conclusiones extraídas de la creación de este *framework* pueden extrapolarse a otros sistemas en los que concurren los mismos condicionantes tecnológicos: (1) conceptos del dominio diferenciados que faciliten el desarrollo de un DSL, (2) existencia de herramientas comerciales para la generación de código, y (3), modelos reutilizables como resultado de aplicar ingeniería de dominio o enfoques de líneas de productos. Tal es el caso de dominios como el robótico, el de visión o los sistemas de control.

Como línea futura, actualmente se está trabajando en la inclusión en el *framework* de otras plataformas específicas (tecnologías domóticas como LonWorks) y la ejecución de este *framework* con herramientas interoperables. También se está mejorando la herramienta de trazabilidad en aspectos como la inclusión de características avanzadas de filtrado, estadísticas de la información de los elementos que se generan en cada nivel, análisis de los elementos de interés comercial (un uso típico de esto es encontrar los elementos que no son requeridos por el sistema, por ejemplo, una característica que no estaba descrito en los requisitos) y, por último, en la integración de la herramienta en entornos distintos a los de Eclipse.

Además, con el fin de valorar la calidad del sistema domótico obtenido, se ha planteado continuar el trabajo iniciado en esta Tesis incorporando un conjunto de métricas. Estas métricas se aplicarán a todo el ciclo de desarrollo permitiendo descubrir y corregir problemas potenciales teniendo en cuenta factores de calidad como, por ejemplo, la facilidad de uso o el mantenimiento.

También es de interés la consideración de otras herramientas que den soporte a la generación de código en KNX/EIB, como el BCU SDK de la Universidad de Viena (<http://www.auto.tuwien.ac.at/~mkoegler/index.php/bcusdk>). Esta herramienta es gratuita y permite escribir código personalizado para los dispositivos KNX OEM¹ o de desarrollo propio. Con esa alternativa se podría evitar la etapa de catalogar las funciones de los dispositivos comerciales (para el refinamiento de componentes).

Conclusions

In this thesis we have introduced a framework for developing HA systems by following a model driven approach. The development of this kind of systems is a time-consuming endeavor that could only be afforded by experienced developers with programming and execution-platform expertise. Our framework allows low-experienced developers to create HA system descriptions by means of a domain specific language and a set of transformations to generate executable code. These features enable the development of HA applications with improved quality both of the process and of the software artifacts obtained.

The use of new Software Engineering techniques into domains traditionally not subject of this study, such as the HA domain, is very promising. Although the MDE approach is not new, there are no integrated proposals for the development of HA systems that cover the entire process. None of them takes into account the advantage of reusing the infrastructure already provided by HA manufacturer.

It may be difficult and laborious to create a DSL from scratch. Nevertheless, the benefits are greater than liabilities in some domains. Creating a new DSL (with tools to support it) can be worthwhile if the language allows a particular type of problems to be expressed more clearly than pre-existing languages (such as UML) would do. In MDE the use of domain-specific languages is the best option for HA systems where the concepts themselves are well defined. In this way, it allows developers with some experience in HA to create descriptions of HA systems using visual notations that can be automatically transformed into executable code.

For the transformations, graph grammar techniques are a good choice given the graphic nature of both the transformations and the models. But the need to work with an integrated tool within Eclipse makes more interesting the use of hybrid declarative/imperative languages.

Our approach makes it possible to deal with the reuse of model fragments from the requirements elicitation phase. It provides a requirements catalogue that can be reused from system to system development. This catalogue has served as a basis to develop an extensible approach that allows developers to reuse DSL-model fragments and hence to incorporate as a partial solution that will be integrated into complete DSL models. This helps modelers to improve the quality of their models and avoid the necessity of modeling recurrent requirements. A set of experiments to help provide an empirical evaluation of the proposal has been given. Thus, the survey demonstrates that the reuse of DSL-based models through generic requirements is really effective. The most encouraging lesson from our work is the opportunity to increase the level of model reuse through the combination of DSLs and generic requirements. It is the synergy between the involved elements in the process which is likely to bring the most significant benefits of reuse in MDD.

Furthermore, both the reuse and traceability capabilities provide our framework with the basic characteristics desirable in any development, improving quality and reducing the resulting software development costs.

All the lessons learned from our experience can be of interest in other domains. For example, since the HA domain is a specific case of reactive systems, these same lessons can be extrapolated to those that attend the same technological conditions: (1) the existence of a substrate for the definition of a DSL, (2) the possibility of reusing models from a software product-line perspective, and (3) the availability of commercial tools for making easier and higher code generation.

Our work is now focused on the consideration of other execution platforms and the interoperability of our framework with other tools (such as an animator to validate the models before final deployment). In addition, work is currently underway to improve the traceability tool to facilitate, for instance, advanced filtering features, statistical information of elements generated in each level, analysis of orphan elements (a typical use of this is to find elements that are not required by the system, e.g. a feature that was not described in the requirements) and, lastly, the integration of the tool in environments other than Eclipse.

Also of interest is the consideration of the BCU SDK tools from the University of Vienna [40] which offer an alternative way to tackle the code generation for the KNX/EIB platform. These free tools could facilitate the code generation as they allow writing customized code for OEM KNX¹ devices. Thus the need of cataloging commercial devices by functionality could be avoided.

Bibliografía

- AENOR - Asociación Española de Normalización y Certificación. Available at: <http://www.aenor.es/aenor/inicio/home/home.asp>.
- Larousse, 2011. *Gran diccionario de la lengua española Larousse*, Spes.
- MDA, 2012, Available at: <http://www.omg.org/mda/>.
- Bézivin, J., On the Unification Power of Models. En *Software and System Modeling*. Available at: <http://www.sciences.univ-nantes.fr/lina/atl/www/papers/OnTheUnificationPowerOfModels.pdf>.
- Demir, A., 2006. Comparison of model-driven architecture and software factories in the context of model-driven development. En *Fourth and Third International Workshop on Model-Based Development of Computer-Based Systems and Model-Based Methodologies for Pervasive and Embedded Software, 2006. MBD/MOMPES 2006*. pp.79–83.
- Espinoza, P.P.R., 2011. CONVERGENCIA DE MEDIOS. *CONVERGENCIA DE MEDIOS | DIDÁCTICA DE LA EDUCOMUNICACIÓN*. Available at: <http://es-educomunicacion.blogspot.com/2010/04/tarea-6-convergenencia-de-medios.html>.
- Favre, J., 2004. Towards a basic theory to model model driven engineering. IN *WORKSHOP ON SOFTWARE MODEL ENGINEERING, WISME 2004, JOINT EVENT WITH UML2004*. Available at: <http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.86.1717>.
- Jimenez, M. et al., 2009a. Habitation: A Domain-Specific Language for Home Automation. *IEEE Software*, 26(4), pp.30–38.
- Jiménez, M. et al., A tool for facilitating the teaching of smart home applications. *Computer Applications in Engineering Education*. Available at: <http://onlinelibrary.wiley.com/doi/10.1002/cae.20521/abstract>.
- Jiménez, M, 2009. *Desarrollo de Sistemas Domóticos Utilizando un Enfoque Dirigido por Modelos*. Tesis Doctoral. Universidad Politécnica de Cartagena. 2008.
- Kemp, N., 2011. *Home Automation Application Guide: Home Automation Applications for Convenience, Comfort and Peace of Mind*, CreateSpace.
- Kruchten, P., 2004. *The rational unified process: an introduction*, Addison-Wesley Professional.
- Mellor, S.J. et al., 2002. Model-Driven Architecture. En J.-M. Bruel & Z. Bellahsene, eds. *Advances in Object-Oriented Information Systems*. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 290–297. Available at: <http://www.springerlink.com/content/28bucqgq68qkrnke/>.

- Mernik, M., Heering, J. & Sloane, A.M., 2005. When and how to develop domain-specific languages. *ACM Comput. Surv.*, 37(4), pp.316–344.
- Métrica Versión 3, Ministerio de Administraciones Públicas. (España), C.S. de I., 2001.
- Miller, F.P., Vandome, A.F. & John, M., 2010. *Model-Driven Engineering*, VDM Verlag Dr. Mueller e.K.
- SOLÉ MORO, M^a LUISA, *CONSUMIDORES-DEL-SIGLO-XXI*, Available at: <http://www.uniliber.com/titulo/CONSUMIDORES-DEL-SIGLO-XXI.html>.
- Sanchez, P. et al., 2011. Introducing Safety Requirements Traceability Support in Model-Driven Development of Robotic Applications. *IEEE Transactions on Computers*, 60(8), pp.1059–1071.
- Sierra, E.J., 2009. *Del Hogar Digital a la Casa Red*, Lulu.com.
- Sillitti, A., 2010. *Agile Processes in Software Engineering and Extreme Programming: 11th International Conference, XP 2010, Trondheim, Norway, June 1-4, 2010, Proceedings*, Springer.
- Sommerville, I., 2005. *Ingeniería del software*, Pearson Educación.
- Sztipanovits, J., 2005. Model integrated computing: foundations and applications. En *Engineering of Computer-Based Systems, 2005. ECBS '05. 12th IEEE International Conference and Workshops on the*. Engineering of Computer-Based Systems, 2005. ECBS '05. 12th IEEE International Conference and Workshops on the. IEEE.
- Sánchez, P. et al., 2011. A framework for developing home automation systems: From requirements to code. *Journal of Systems and Software*, 84(6), pp.1008–1021.
- Tolvanen, J.-P., 2011. MetaEdit+: domain-specific modeling environment for product lines. En *Proceedings of the 15th International Software Product Line Conference, Volume 2. SPLC '11*. New York, NY, USA: ACM, pp. 51:1–51:2. Available at: <http://doi.acm.org/10.1145/2019136.2019195> [Accedido febrero 10, 2012].
- Jiang, T. and Yun, W. "Research on formalization of Domain-Specific Metamodeling Language based on first-order logic", *Computer Science and Automation Engineering (CSAE), 2011 IEEE International Conference on*, vol. 4 n° 10, pp. 170 – 174, 2011.
- Clark, T., Sammut, P. Willans, J.. *Applied Metamodeling a Foundation for Language Driven Development*. CETEVA, 2008, Available at <http://www.ceteva.com>.
- Czarnecki, K. and Helsen, S. "Classification of Model Transformation Approaches". OOPSLA 03 Workshop on Generative Techniques in the Context of Model-Driven Architectures, 2003.

- Object Management Group (OMG), 2012. Meta Object Facility (MOF) Core Specification, Available at: <http://www.omg.org/docs/formal/06-01-01.pdf>
- Favre, J. Foundations of Meta-Pyramids: Languages vs Metamodels - Episode II: Story of Thotus the Baboon1, in Language Engineering for Model-Driven Software Development, 2004
- Mens T. and vanGorp, P. . A taxonomy of model transformation, Electronic Notes in Theoretical Computer Science 152, pp. 125–142 ,2006.
- Rozenberg, G.. Handbook of Graph Grammars and Computing by Graph Transformation, 1sted, World Scientific, Netherlands, 1997.
- JET. 2012. Eclipse Consortium, Java Emitter Templates (JET). Available at <http://www.eclipse.org/modeling/m2g/?project=jet>.
- Rumbaugh, J., Blaha, M., Premerlani, W., Eddy, F., Lorensen, W. 2005. Object-Oriented Modeling and Design. Prentice Hall, New York. p. 314.
- IEEE. 830-1984 IEEE Guide to Software Requirements Specifications. Volume4. 1999. The Institute of Electrical and Electronics Engineers. Inc. IEEE Software Engineering Stds. Collection.
- Loucopoulos, P. and Karakostas, V.. System Requirements Engineering. McGraw-Hill, 1995.
- Sommerville, I. and P. Sawyer. Requirimentes Engineering: A Good Practice Guide. John Wiley and Sons, 1997.
- Cybulsky, J. and K. Reed. Requirements Clasification and Reuse: Crossing Domains Boundaries. 6th Intl. Conf. On Software Reuse (ICSR'2000). Viena, Austria. 2000.
- Laguna, M. ,O. López and Y. Crespo. Reuse, Standarization, and Transformation of Requirements. 8th Intl. conf. on Software Reuse (ICSR 2004). Madrid. 2004.
- Robertson S. and J. Roberstson. Mastering the Requirements Process. 2nd. Ed. New York. 2006.
- Monzón A. and J. Dueñas. Experience-based Approach to Requirements Reuse in Product Families with DOORs. Intl. Workshop on Requirements Reuse In System Family Enginnering. Madrid. 2004.
- Cerón, R., J. Dueñas, E. Serrano and R. Capilla. A Meta-Model for Requirements Engineering in System Family Context for Software Process Improvement Using CMMI. Intl. Conf. on Product Focused Software Process Improvement (PROFES 2005). Finland. 2005.
- Olsen G.K. and J. Oldevik. Scenarios of Traceability in Model to text Transformations, in: 3rd European Conference on Model-Driven Architecture Foundations and Applications (ECMDA'07), Israel, 2007.

- Kolovos, D., R. Paige and F. Polack. On-demand merging of traceability links with models, in: Proceedings of the 2nd EC-MDA Workshop on Traceability, Bilbao, Spain, 2006.
- Ramesh B. and M. Jarke. Toward Reference Models for Requirements Traceability. IEEE Transactions on Software Engineering, volume 27, issue 1, pp.58-93, 2001.
- Behrens. T. 2007. Never Without a trace: Practical advice on implementing traceability. Available at <http://www.ibm.com/developerworks/rational/library/feb07/behrens>.
- Almeida, J., P. van Eck and M. Iacob. Requirements Traceability and Transformation Conformance in Model-Driven Development. Proceedings of the 10th IEEE International Enterprise Distributed Object Computing Conference (EDOC'06), IEEE Computer Society Press, pp. 355–366, 2006.
- Cleland-Huang, J. , J. Huffman, J. Domel. Model-based traceability. Workshop on Traceability in Emerging Forms of Software Engineering (ICSE 2009). pp. 6-10, 2009.
- Gotel O. and C. Finkelstein. An analysis of the requirements traceability problem. In: Proceedings of the first international conference on requirements engineering. Uthrecht, The Netherlands; 1994.
- Melby. S. Traceability in Model Driven Engineering. Master Thesis. University Of Oslo, Norway, <http://urn.nb.no/URN:NBN:no-18721>, Nov. 2007.
- Fernandez, E. Insfrán, S. Abrahao. Towards a Usability Evaluation Process for Model Driven Web Development. Proceedings of the 2nd International Workshop on the Interplay between Usability Evaluation and Software Development, 2009.
- Piattini, M., F. Garcia, J. Garzás, M. Genero. Medición y estimación del software: Técnicas y métodos para mejorar la calidad y la productividad. Editorial Ra-Ma. 2008.

ANEXO A: CARTAS DE ACEPTACIÓN

Habitation: A Domain-Specific Language for Home Automation

IEEE Software, SWSI-2008-11-0248.R2
manuscript type: Special issue on Domain-Specific Languages & Modeling
- Jul/Aug 2009
"HABitATION: a Domain Specific Language for Home Automation"

24-Mar-2009

Dear Mr. Jiménez,

Congratulations! Your above referenced manuscript has been officially accepted for publication in IEEE Software's Special issue on Domain-Specific Languages & Modeling - Jul/Aug 2009, subject to final editing for English style, clarity, and organization.

Once your article is scheduled for publication, we will assign it to a professional editor. The editor will contact you directly to discuss any recommended changes that will enhance the article's presentation, as detailed in our editing philosophy and author guidelines (<http://www.computer.org/mc/software/author.htm>). The final article will be the result of a collaborative process between you and the IEEE Software staff editors. Please note that IEEE Software reserves the right to change the title of any paper that is accepted for publication.

Also, please understand that the editing process won't begin until we schedule your article for a specific issue and assign a staff editor. At that point, the staff editor will contact you. Because we work on four magazines simultaneously, we typically don't get a chance to work in advance. We will try our best to give you a week to review the first edit and several days to review the layout.

If you know of any scheduling conflicts in advance or if you have any questions, feel free to contact me via email or phone (+1 714 821 8380).

Sincerely,
Dale Strok

Senior Lead Editor
IEEE Software
10662 Los Vaqueros Circle
Los Alamitos, CA 90720
+1 714 821 8380---phone
+1 714 821 4010---fax
<http://computer.org/software>

A Tool for Facilitating the Teaching of Smart Home Applications

De: onbehalf+iskander@spectra.eng.hawaii.edu@manuscriptcentral.com
[mailto:onbehalf+iskander@spectra.eng.hawaii.edu@manuscriptcentral.com] En
nombre de iskander@spectra.eng.hawaii.edu
Enviado el: miércoles, 22 de diciembre de 2010 2:56
Para: manuel.jimenez@upct.es; pedro.sanchez@upct.es
Asunto: CAE-10-028.R1 - Decision

12/21/2010

Dear Dr. Jimenez,

It is a pleasure to accept your manuscript entitled "A tool for facilitating the teaching of smart home applications" in its current form for publication in Computer Applications in Engineering Education.

A signed copyright transfer agreement is needed for publication. If you have not already provided one you should do so immediately. You can access the copyright transfer agreement at www.wiley.com/go/ctaaus.

Please print the form, complete both sides and e-mail to the Production Department at lespinet@wiley.com.

You will receive your typeset proofs in due course.

Thank you for your contribution.

Sincerely,

Prof. Magdy F. Iskander
Director, Hawaii Center for Advanced Communications
College of Engineering
University of Hawaii at Manoa
2540 Dole Street, Holmes Hall 483
Honolulu, HI 96822
808-956-3434
iskander@spectra.eng.hawaii.edu

Introducing Safety Requirements Traceability Support in Model-Driven Development of Robotic Applications

RE: TCSI-2009-06-0305.R1, "Introducing Safety Requirements
Traceability Support in Model-Driven Development of Robotic
Applications"
Manuscript Type: Special Issue on Science of Design for Safety
Critical Systems

18-Mar-2010

Dear Dr. SANCHEZ,

We have completed the review process of the above-referenced paper,
submitted to the IEEE Transactions on Computers.

I am pleased to inform you that, based on the reviews, Associate
Editor Dr. Safety Critical Systems Guest Editors has recommended to
the Editor-in-Chief that your paper be published as a regular paper in
an upcoming issue of the IEEE Transactions on Computers with no
further changes.

As of 1 Feb. 2010, PDF files of final, accepted papers will be
downloaded from ScholarOne Manuscripts and used for preprints. All
editable versions must match this final accepted version downloaded
from ScholarOne. Please be advised that any subsequent PDF files will
not be used for production.

If for any reason you would prefer not to have your paper posted as a
Preprint, please let me know within 3 business days of this notice;
otherwise, not responding indicates your acceptance of our policy. If
you opt out of preprints, please re-name your directory on the FTP
site following the convention, TCSI-2009-06-
0305.R1_SANCHEZ_DO_NOT_POST_ONLINE.

Thank you for your contribution to Transactions on Computers. We look
forward to seeing your final files soon.

Joel T. Luber, on behalf of:
Fabrizio Lombardi, EIC
Transactions on Computers
tc@computer.org

A framework for developing home automation systems: from requirements to code

De: Author Services <support@elsevier.com>
Fecha: 20 de febrero de 2011 08:51:41 GMT+01:00
Para: pedro.sanchez@upct.es
Asunto: Article tracking [JSS_8657] - Online publication complete

Article title: A framework for developing home automation systems:
from requirements to code

Reference: JSS8657

Journal title: The Journal of Systems & Software

Corresponding author: Dr. Pedro Sánchez

First author: Dr. Pedro Sánchez

Online publication complete: 18-FEB-2011

DOI information: 10.1016/j.jss.2011.01.052

Dear Dr. Sánchez,

The final corrections to your proof have been made. Further
corrections cannot be made. Your article is now published online at:

<http://dx.doi.org/10.1016/j.jss.2011.01.052>

Please note access to the full text of this article will depend on
your personal or institutional entitlements.

Until the bibliographic details (including volume and page numbering)
are assigned, your article is citable by DOI.

You will be automatically notified by e-mail once the full
bibliographic details are available.

Corresponding authors who wish to receive hard-copy offprints are
reminded to return their completed offprint order form as soon as
possible; otherwise, authors will be charged an additional late order
fee varying from 15-50% post publication depending on the journal.

To track the progress of your article via our article tracking service
please visit <http://authors.elsevier.com/MyHome.html>.

Yours sincerely,
Elsevier Author Support
<http://support.elsevier.com>

ANEXO B: ARTÍCULOS

Habitation: A Domain-Specific Language for Home Automation

**Manuel Jiménez, Francisca Rosique, Pedro Sánchez, Bárbara Álvarez,
and Andrés Iborra,** *Technical University of Cartagena*

Combining a domain-specific language with a model-driven approach can enhance the quality and portability of home automation systems.

The appearance of model-driven engineering (MDE)¹ has invigorated research on domain-specific languages (DSLs)² and automatic code generation. MDE uses models to build software, thereby displacing source code as the development process's main feature. DSLs provide easy, intuitive descriptions of the system using graphic models. In this new context, DSLs facilitate work in the first design stages. In addition, MDE helps reduce DSL development costs.^{3,4} It therefore represents a synergistic union that can significantly improve software development.

MDE lets developers create tools for managing reactive systems much more effectively.^{5,6} These tools are more interoperable than traditional tools and easier to maintain owing to certain MDE qualities, such as a higher abstraction level. Home automation is one example of MDE's application in reactive systems. Home automation systems can interact with their environments, offering management of energy, security, communications, and comfort.⁷ Such systems are currently developed using low-level procedures and without a methodology that allows platform-independent inclusion of the system requirements. Home automation application developers must therefore have a high level of specialization. In addition, only minimal reuse of artifacts is possible.

Specific languages that allow platform-independent capture of requirements are practically nonexistent in the home automation field. MDE-based proposals for home automation^{6,8} use modeling languages (such as UML⁹), which

aren't very intuitive and are far removed from the home automation sphere. For example, UML includes hundreds of elements, but only a few of them are directly relevant to software design. Even using profiles, models would be complex (plenty of tags, stereotypes, and so on). Other proposals correspond to platform-dependent commercial tools; the two best-known are Engineering Tool Software (ETS) and LonMaker, which are specific to the KNX/EIB (European Installation Bus) and LonWorks platforms, respectively.

In light of all this, we introduce Habitation (derived from *development of home automation applications using a model-driven approach*), a new methodology to tackle the complete life cycle of home automation system design. Habitation combines a model-driven approach with DSLs to support these applications' definition. We also offer a platform- and technology-independent graphical tool that uses domain-specific abstractions.

Figure 1. A graphic display of the catalog view. This view's elements include (a) category; (b) links between categories (green) and between functional units and categories (blue); (c) functional unit; (d) services, with the signs indicating whether the service is provided or required (right or left arrow) and whether it's a hardware or software service (red or blue); (e) parameter definition; and (f) scene.

The Home Automation DSL

Home automation application developers mainly use a software tool provided by the device manufacturer (in the case of proprietary systems) or the associations that support the technology (in the case of standardized systems). These tools are usually platform-dependent, code-generation-oriented, integrated environments that do little to raise the abstraction level. Moreover, the concrete syntax they use is rarely intuitive, so users require specialized training and can work only in the solution's immediate context.

The entire application development process is performed by a domain expert who collates the customer's requirements for an installation (elements to be integrated, services required, selection of a concrete technology, and so on) on the basis of the expert's own experience. This expert deploys the devices and then programs them (using a platform-specific tool) to achieve the desired functionality. This manner of working makes it difficult to achieve some of the desired attributes of software systems, such as interoperability, flexibility, reuse, and productivity.

To resolve these shortcomings, we combine a specific visual language with an MDE approach. Our main objective in defining this language is to let designers describe home automation systems using only domain concepts. In this sense, our DSL facilitates the requirements-specification phase visually and intuitively. So, the first constraint is to provide a visual language that's concise and common to the different platforms.

Any home automation system incorporates several elements (*functional units*) that are in all the technologies and standards proper to the domain. These employ different architectures and protocols,

but they're identical in capability. To encourage reuse of these functional units and to avoid repeatedly defining the same unit for each application (including several times in a single application), we used a catalog of reusable elements. Once you define such a catalog, you can use it in any application. Functional units have some services through which they can interact with other units. Many of these services are repeated among the functional units, so we created a catalog of services with service definitions that we can reuse in any functional unit.

For this reason, we differentiate two approaches to DSLs. In one, the DSL's purpose is to develop applications, and the user is a developer who might be familiar with the field but isn't necessarily an expert. In the other, the purpose is to develop and implement possible catalog upgrades, and the user should be an expert in the field.

The Catalog View

This view lets the home automation expert model the catalog of functional units and services that developers will later use to create home automation applications. Figure 1 shows the main primitives for modeling a catalog:

- A *category* is a specialization of a catalog element.
- *Links* can be between categories or between functional units and categories.
- A *functional unit* is the smallest element into which a home automation device can be divided. It includes an icon, a name, and the services provided or implemented.
- A *service definition* has a signature that includes the service name and its arguments. The service sign indicates whether the service is

Figure 2. A catalog of functional units and their categories. Final-Passive functional units represent nonprogrammable elements; Controller functional units represent programmable elements.

provided or required (right or left arrow) and whether it's a hardware or software service (red or blue). A service catalog serves as a compartment containing the service definitions.

- A *parameter definition* indicates a functional unit's parameters.
- A *scene* is a specialization of the functional unit. We define the steps that constitute a scene later.

These elements are available in the developed tool's palette.

Figure 2 shows a snapshot of a catalog, which includes categories and functional units that any home automation application developer would use. We'll upgrade and enlarge this catalog by incorporating new functional-unit definitions inside the existing categories. The Final-Passive functional units represent unprogrammable elements (for example, lights and push buttons). The Controller functional units represent programmable elements.

The Application View

This view is used by the developer who designs new applications, who needn't be a home automation expert. The developer can use the catalog to specify an application using these primitives:

- Developers configure *instances of functional units*, which are defined in the catalog, by adding the necessary values to their parameters.

- The *links between functional units* indicate, through services, how these units will interact with the rest of the system. The links can act as channels when a functional unit involved is passive (as such, it's modeled as a hardware-level connection) or as a normal link when neither unit is passive.

- Developers can use *scenes* to configure the sequential execution of several services from functional units within a single action. For example, a developer could define a "Presentation" scene using three steps: lower the blinds, dim the lights, and lower the projection screen. The user could push a button to trigger this scene.

Figure 3 shows these elements, which are included in the developed tool's palette.

Model-Driven Methodology and Tools

Our methodology uses the Object Management Group's model-driven architecture (MDA),¹⁰ which organizes software development in three layers:

- a *computation-independent model* (CIM), which in our case represents the syntax and part of the semantics of the defined DSL;
- a *platform-independent model* (PIM), which in our methodology is a simplification of the UML metamodel for reactive systems,⁵ and consid-

ers components, activities, and state chart diagrams; and

- a *platform-specific model* (PSM), for which we've defined a metamodel for the KNX/EIB home automation technology. This metamodel considers the domain object model used by ETS.

In the CIM layer (see Figure 4), the developer elicits requirements through the DSL. Models from this level are automatically transformed into architectural components in the PIM layer. Our tool then transforms the components into executable PSMs for each platform.

This methodology requires that we use the DSL in the first development phase (CIM) so that the user can interact easily with the tool, relying on the methodology's underlying precision. The PIM level is a junction point for different reactive systems (wireless sensor networks, robotic systems, artificial vision, and so on). Consequently, the elements of home automation systems designed in this manner can be integrated as components of a more complex reactive system.

The tool we developed to support our methodology uses the Eclipse (www.eclipse.org) development environment. Eclipse provides a working framework in which the user can manage models. It incorporates various MDE-related projects, making it possible to perform modeling, model transformation, verification, graphic environment generation, code generation, and other such tasks.

The Eclipse Modeling Framework is a plug-in in the Eclipse development environment. EMF lets you create model editors and supplies the basis for interoperability with other tools. We used the Eclipse Graphical Modeling Framework (GMF), which automatically generates graphic editors as Eclipse plug-ins from models.

Our DSL tool has three parts:

- a drawing area in which to build graphic models for the catalog and applications,
- a graphic palette containing the elements (see Figures 1 and 3) that can be dragged to the drawing area, and
- an area in which the available properties (attributes, parameters, and so on) are displayed and can be modified for the selected element.

The tool, which lets us create logical models that describe applications in terms of functional units and links between their services, is now fully operational. A demonstration of its use in our case study example is available at <http://hdl.handle.net/>

Figure 3. A graphic display of the application view. (a) An instance of a functional unit includes the unit's parameters and their values. (b) Links between functional units can act as channels (red line) or other link types (discontinuous green line with end points). In each link, the top and bottom labels indicate services that participate in the link; the center label is the service's definition. (c) Scenes contain the steps to be performed. A step shows the service to be performed and the icon of the functional unit to which it belongs.

10317/854. The CIM metamodel supports an additional floor-plan view, but its implementation in the tool is still under development.

The transformations between the CIM and PIM layers are completely defined using a graph-grammar-based approach¹¹—in particular, the EMF Model Transformation (EMT) plug-in.¹² Because models are usually represented by graphs, graph grammar is more attractive than other approaches. For instance, transformation rules expressed through graphs are easier to understand and trace.

Transformation is expressed with rules. Each rule has a left-hand side (LHS) and a right-hand side (RHS), both of which are graphs. A rule might also have a negative application condition (NAC), which must not be satisfied to apply it. To apply a rule to a host graph (the graph to be transformed), a subgraph isomorphism from the LHS to the host graph must exist. After the application, there must be a subgraph isomorphism from the RHS to the result graph.

Consider the model-to-model transformation (from DSL to PIM) in Figure 5. Figure 5a represents a push button (PB-1) that switches a light (LO-1) on and off. Elements SWI-1 and SWO-1 symbolize the controllers providing the desired functionality. Figure 5b shows a graph transformation rule. The rule states that when a service (LHS) is found, it must be transformed into ports, interfaces, and services of the target component model (RHS). However, this rule isn't applied if the transformation has

Figure 4. The proposed methodology for developing home automation applications. The CIM level captures user requirements using the defined domain-specific languages (DSLs). The PIM level considers a UML-like component model. At the PSM level, we provide models for different specific platforms and several code generation strategies.

already been performed (NAC). Applying all the rules results in a component model (see Figure 5c). A complete description of the CIM-to-PIM graph transformation rules and the considered metamodels is available elsewhere.¹³

To generate code, the developer must first select a target platform. Doing this involves two key considerations:

- The technology must be supported by international standards.
- The tools for programming the devices must be available and able to be interfaced externally.

The two leading home automation technologies—

KNX and LonWorks—fulfill these requirements. Because our research group has wide experience in KNX, we selected this technology for the first platform-specific infrastructure. Currently, the rules for transforming PIM models into PSM models (conforming to the defined KNX/EIB meta-model) are informally defined. We're working to formalize these rules using the graph-grammar notation. PSM models are independent of specific commercial tools and serve as a source for model-to-text transformations. To implement these transformations, we chose the Java Emitter Template tool (JET; www.eclipse.org/modeling/m2t/?project=jet) and the ITTools plug-in. This plug-in lets us interface with the manufacturer

Figure 5. Obtaining a component diagram from a DSL model. (a) A DSL example with the source model at the CIM level. (b) A graph transformation rule for a transform from the CIM to the PIM level. Black indicates a catalog view (DSL) instance, red indicates a target metamodel (PIM) instance, blue indicates an application view (DSL) instance, and green indicates a transformation metamodel instance. (c) A component model diagram with the target model at the PIM level.

environment (ETS) using the VBScript programming language. In this way, we promote reuse of platform-specific tools.

Case Study

Our sample case study involves a system that controls and manages a meeting room used for meetings, seminars, and presentations of various kinds (see Figure 6).¹³ This case study has let us validate the DSL's functionality in a real application and establish a starting point from which to apply our methodology. It aims to achieve various objectives regarding energy, security, comfort, and communications.

We used the DSL application view to formally display the system requirements. The complete specification is given by the model's graphical view plus the parameterization of the corresponding properties. Links between functional units establish a kind of activity diagram that starts from the events triggered at the input units. Parameters set up the controllers' internal behavior.

Figure 6 shows the application model for light-

ing management, including the deployment layout and some correspondences with the DSL elements. Two push buttons control six lighting points in the room. One push button (PB-1) switches and dims the lights next to the meeting room window (LDM-1 to LDM-4). Another push button (PB-2) controls the lights next to the door (LDM-5 and LDM-6). To model this behavior, we connect each push button to a dimming input controller (DMI-1 and DMI-2) using a channel link (red lines) that binds required (PBactivated) and provided (DMI-activated) services (both services must be instances of the same service definition). At the same time, lights are linked to their controllers (DMO-1 to DMO-6), which switch and dim the lights. Finally, controller services are associated with logical links (dashed green lines).

To achieve energy saving and comfort, the system activates a power-off function when it detects no presence in the room after five minutes. Presence detectors (PIR3-1 and PIR3-2) are interconnected through channels to the functional units acting as controllers (SWI-1 and SWI-2). These

Figure 6. The home automation system in our case study: (a) floor plan and element layout and (b) lighting model in the Habitation DSL. Dashed lines show correspondences between the physical devices and the Final-Passive functional units. The remaining DSL elements are integrated either as part of these devices or as separate controllers.

controllers call the TMtempIn service from the timer (TM-1) every time the system detects a presence. So, the timer switches the lights on when the system detects a presence and switches them off after 300 seconds of detecting no presence. It's possible to use presence detectors simultaneously for both lighting and security management. The functional-unit icons suggest their meanings. Execution platforms (KNX/EIB, LonWorks, and so on) usually integrate controllers (in the case study example, DMI-1, DMO-1, and so on) into devices following a specific criterion. So, Figure 6 doesn't include correspondences between these controllers and floor-plan devices.

Evaluation

Software products should be evaluated for each relevant quality factor using widely accepted metrics.¹⁴ To validate our proposal and our DSL's possible benefits, we conducted an experiment involving a group of students in an electronic-engineering master's course on home automation.

We offered participants, none of whom had previous knowledge of home automation technology, three training sessions before beginning the evaluation. The first involved training in the home automation field. The two subsequent sessions provided training in the use of a commercial tool and in the use of the DSL. We then presented participants with a case study in which they were to use both tools.

The experiment mainly concerned usability under specific conditions.¹⁵ It evaluated six usability quality factors: ease of understanding, ease of learning, operability, flexibility, accordance, and attractiveness. For each quality factor, we asked participants to perform an action using the tools and then complete a questionnaire rating their experience (using a 1- to 5-point scale, where 5 is the highest quality rating). We also tracked the time needed to complete the actions and to respond to the questionnaire. We obtained final valuations for each factor using an arithmetic mean of the results of each questionnaire. Table 1 lists the results.

Table 1

Usability results from our case study

Quality factor	Home automation DSL			Commercial tool		
	Score (1 to 5)	Time needed (min.)		Score (1 to 5)	Time needed (min.)	
		Action	Questionnaire		Action	Questionnaire
Ease of understanding	3.6	20.0	5	2.4	24.6	4
Ease of learning	4.2	30.0	3	2.0	90.0	8
Operability	3.6	20.0	6	3.0	25.0	3
Flexibility	2.8	36.8	6	3.6	23.4	4
Accordance	3.8	12.2	5	3.4	10.0	3
Attractiveness	4.0	10.0	3	2.0	20.0	4

Study participants mostly rated the DSL tool higher than the commercial tool (ETS). The exception was flexibility. The largest differences were in ease of learning, ease of understanding, and attractiveness. For example, to questions such as, “Do you need help to remember the concepts represented by each primitive in the palette/tool bar?” most students answered “not at all” (score 5) in the DSL questionnaire. Fewer students did so for the commercial tool.

We’re completing the code-generation implementation for ETS. We’re also working to integrate other home automation platforms and advanced capabilities, such as requirements traceability, in the process.

Incorporating the GMF plug-in offers multiple possibilities for managing models. However, the training time required to effectively use it is long. With this in mind, we’re exploring other modeling tools that require less training time, such as MetaEdit+, which offers a fully integrated modeling, metamodeling, and code generation environment.¹⁶

Acknowledgments

The Spanish Interministerial Commission of Science and Technology’s MEDWSA (a conceptual and technological framework for the development of reactive software systems) project (TIN2006-15175-C05-02) and the Technical University of Cartagena partially supported this work.

References

1. B. Selic, “The Pragmatics of Model-Driven Development,” *IEEE Software*, vol. 20, no. 5, 2003, pp. 46–51.
2. M. Mernik et al., “When and How to Develop Domain-Specific Languages,” *ACM Computing Surveys*, vol. 37, no. 4, 2005, pp. 316–344.

About the Authors

Manuel Jiménez is an associate professor of industrial electronics at the Technical University of Cartagena and is a member of the university’s DSIE (Division of Systems and Electronic Engineering) research group. His research interests include electronics and model-driven engineering for reactive systems. Jiménez has a PhD in computer science from the Technical University of Cartagena. Contact him at manuel.jimenez@upct.es.

Francisca Rosique is an assistant professor and a PhD student in computer science at the Technical University of Cartagena and a member of the university’s DSIE (Division of Systems and Electronic Engineering) research group. Her research interests include model-driven engineering and home automation systems. Rosique has a master’s in telecommunication engineering from the Technical University of Cartagena. Contact her at paqui.rosique@upct.es.

Pedro Sánchez is an associate professor of computer science at the Technical University of Cartagena and a member of the university’s DSIE (Division of Systems and Electronic Engineering) research group. His research interests include model-driven engineering for real-time systems. Sánchez has a PhD in computer science from the Technical University of Valencia. Contact him at pedro.sanchez@upct.es.

Bárbara Álvarez is an associate professor in computer science at the Technical University of Cartagena and a member of the university’s DSIE (Division of Systems and Electronic Engineering) research group. Her research interests include real-time systems and software architectures for teleoperation. Álvarez has a PhD in telecommunication engineering from the Technical University of Madrid. Contact her at balvarez@upct.es.

Andrés Iborra is full professor and head of the Electronics Technology Department at the Technical University of Cartagena and a member of the university’s DSIE (Division of Systems and Electronic Engineering) research group. His research interests include computer vision and robotics. Iborra has a PhD in industrial engineering from the Technical University of Madrid. Contact him at andres.iborra@upct.es.

3. T. Clark et al., *Applied Metamodeling: A Foundation for Language Driven Development*, Ceteva, 2008; <http://itcentre.tvu.ac.uk/~clark/Publications.html>.
4. J.P. Tolvanen and S. Kelly, "Defining Domain-Specific Modeling Languages to Automate Product Derivation: Collected Experiences," *Proc. 9th Int'l Conf. Software Product Lines (SPLC 05)*, LNCS 3714, Springer, 2005, pp. 198–209.
5. D. Alonso et al., "V3Studio: A Component-Based Architecture Modeling Language," *Proc. 15th Ann. IEEE Int'l Conf. Workshop Eng. Computer-Based Systems (ECBS 08)*, IEEE Press, 2008, pp. 346–355.
6. M. Voelter and I. Groher, "Product Line Implementation Using Aspect-Oriented and Model-Driven Software Development," *Proc. 11th Int'l Software Product Line Conf. (SPLC 07)*, IEEE CS Press, 2007, pp. 233–242.
7. J.L. Ryan, "Home Automation," *Electronics & Comm. Eng. J.*, vol. 1, no. 4, 1989, pp. 185–192.
8. E. Serral et al., "A Model Driven Development Method for Developing Context-Aware Pervasive Systems," *Proc. 5th Int'l Conf. Ubiquitous Intelligence Computing (UIC 08)*, LNCS 5061, Springer, 2008, pp. 662–676.
9. *Unified Modeling Language (UML) Specification v2.1.2*, Object Management Group, Nov. 2007; www.omg.org/spec/UML/2.1.2.
10. S. Mellor et al., *MDA Distilled*, Addison-Wesley Professional, 2004.
11. G. Rozenberg, *Handbook of Graph Grammars and Computing by Graph Transformation*, World Scientific, 1997.
12. E. Biermann et al., "Tiger EMF Model Transformation Framework (EMT)," 2006; <http://user.cs.tu-berlin.de/~emftrans/papers/userdoc.pdf>.
13. M. Jiménez, "Development of Home Automation Applications Following a Model Driven Approach," PhD thesis, Electronics Technology Dept., Tech. Univ. of Cartagena, 2009; <http://hdl.handle.net/10317/846> (in Spanish).
14. C. Wohlin and P. Runeson, *Experimentation in Software Engineering: An Introduction*, Springer, 2000.
15. A. Seffah et al., "Usability Measurement and Metrics: A Consolidated Model," *Software Quality J.*, vol. 14, no. 2, 2006, pp. 159–178.
16. J.P. Tolvanen and M. Rossi, "MetaEdit+: Defining and Using Domain-Specific Modeling Languages and Code Generators," *Proc. 18th Ann. ACM SIGPLAN Conf. Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA 03)*, ACM Press, 2003, pp. 92–93.

For more information on this or any other computing topic, please visit our Digital Library at www.computer.org/csdl.

CALL FOR ARTICLES

Agility and Architecture— Oil and Water?

PUBLICATION: March/April 2010

SUBMISSION DEADLINE: 17 August 2009

Agile development approaches have had significant impact on industrial software development practices. Nevertheless, there is increasing perplexity about the role and importance of a system's software architecture in agile approaches. Advocates of architecture's vital role in achieving quality goals of large-scale software-intensive systems are skeptical of the scalability of any development approach that does not pay

sufficient attention to architectural aspects, especially in domains like automotive, telecommunication, finance, and medical devices. But agile proponents usually perceive the upfront design and evaluation of architecture as being of little value to a system's customers. This issue intends to separate facts from myths about the necessity, importance, advantages, and disadvantages of coexistence of agile and architectural approaches.

FOR MORE INFORMATION ABOUT THE FOCUS, CONTACT THE GUEST EDITORS:

- Pekka Abrahamsson, Univ. of Helsinki & SINTEF ICT, pekka.abrahamsson@ieee.org
- M. Ali Babar, Lero, malibaba@lero.ie
- Philippe Kruchten, Univ. of British Columbia, pbk@ece.ubc.ca

FOR SUBMISSION DETAILS AND GENERAL AUTHOR GUIDELINES

www.computer.org/software/author.htm or software@computer.org

IEEE
Software

Impact Factor Trend Graph: IEEE SOFTWARE

Click on the "Return to Journal" button to view the full journal information.

***Impact Factor -- see below for calculations**

The journal impact factor is a measure of the frequency with which the "average article" in a journal has been cited in a particular year. The impact factor will help you evaluate a journal's relative importance, especially when you compare it to others in the same field. For more bibliometric data and information on this and other journal titles click on the "Return to Journal" button.

NOTE: Title changes and coverage changes may result in no impact factor for one or more years in the above graph.

2010 Impact Factor

Cites in 2010 to articles published in: 2009 = 85	Number of articles published in: 2009 = 70
2008 = 122	2008 = 67
Sum: 207	Sum: 137

Calculation: $\frac{\text{Cites to recent articles}}{\text{Number of recent articles}} = \frac{207}{137} = \mathbf{1.511}$

2009 Impact Factor

Cites in 2009 to articles published in: 2008 = 134	Number of articles published in: 2008 = 67
2007 = 129	2007 = 62
Sum: 263	Sum: 129

Calculation: $\frac{\text{Cites to recent articles}}{\text{Number of recent articles}} = \frac{263}{129} = \mathbf{2.039}$

2008 Impact Factor

Cites in 2008 to articles published in: 2007 = 107	Number of articles published in: 2007 = 62
2006 = 168	2006 = 69
Sum: 275	Sum: 131

Calculation: $\frac{\text{Cites to recent articles}}{\text{Number of recent articles}} = \frac{275}{131} = \mathbf{2.099}$

2007 Impact Factor

Cites in 2007 to articles published in:	2006 = 90	Number of articles published in:	2006 = 69
	2005 = 81		2005 = 48
	Sum: 171		Sum: 117

Calculation: $\frac{\text{Cites to recent articles}}{\text{Number of recent articles}} = \frac{171}{117} = \mathbf{1.462}$

2006 Impact Factor

Cites in 2006 to articles published in:	2005 = 44	Number of articles published in:	2005 = 48
	2004 = 85		2004 = 55
	Sum: 129		Sum: 103

Calculation: $\frac{\text{Cites to recent articles}}{\text{Number of recent articles}} = \frac{129}{103} = \mathbf{1.252}$

[Acceptable Use Policy](#)
Copyright © 2012 [Thomson Reuters](#).

Published by Thomson Reuters

Rank in Category: IEEE SOFTWARE**Journal Ranking**

For **2010**, the journal **IEEE SOFTWARE** has an Impact Factor of **1.511**.

This table shows the ranking of this journal in its subject categories based on Impact Factor.

Category Name	Total Journals in Category	Journal Rank in Category	Quartile in Category
COMPUTER SCIENCE, SOFTWARE ENGINEERING	99	24	Q1

Category Box Plot

For **2010**, the journal **IEEE SOFTWARE** has an Impact Factor of **1.511**.

This is a box plot of the subject category or categories to which the journal has been assigned. It provides information about the distribution of journals based on Impact Factor values. It shows median, 25th and 75th percentiles, and the extreme values of the distribution.

A Tool for Facilitating the Teaching of Smart Home Applications

MANUEL JIMÉNEZ, PEDRO SÁNCHEZ, FRANCISCA ROSIQUE, BÁRBARA ÁLVAREZ, ANDRÉS IBORRA

DSIE Research Group, Technical University of Cartagena, Campus Muralla del Mar s/n, Cartagena E-30202, Spain

Received 22 February 2010; accepted 21 December 2010

ABSTRACT: This article presents a new tool that can be used to improve the teaching methods of smart home systems development. The main contribution of this tool is that it allows stakeholders to develop home automation (HA) systems using platform-independent graphical elements. Teachers can use this tool to introduce the development of smart home applications and to demonstrate concepts and theories pertaining to this domain. In addition, students can use the tool as an autonomous-learning environment. From the students' point of view, the development of a new system starts with the use of a smart home-specific graphical language, instead of using platform-dependent tools. The article is not only concerned with the description of the tool but also with the methodology to be followed by teachers and students. An example of use is also included illustrating the usefulness of the tool in engineering education. In order to evaluate the benefits of the approach, we conducted a survey amongst students who used the tool. The analysis of the outcome of this survey shows the chance to enable the development of HA applications with improved usability quality factors. ©2011 Wiley Periodicals, Inc. *Comput Appl Eng Educ*; View this article online at wileyonlinelibrary.com; DOI 10.1002/cae.20521

Keywords: home automation; tool for education; graphical language

INTRODUCTION

In the last few years the rapid evolution of communications technologies and electronics has given rise to an increasing demand for smart home systems [1], also known as home automation (HA). The aim of HA applications is to provide homes with a certain degree of “intelligence” and to improve the quality of life of their inhabitants. These systems integrate comfort, energy saving, security, and communications functions. There is a very interesting potential market for HA systems. More and more international research groups are working on this domain [2–4].

A typical HA system includes the implementation of tasks such as: switching on the heating and lighting of a room when somebody enters and switching them off when they leave (comfort and energy saving function); raising and lowering the blinds at sunrise and sunset, respectively (comfort); cutting off the water supply when a water leak is detected (security); cutting off the gas supply when a gas leak or a fire is detected (more security); simulate the presence of inhabitants when nobody is at home by switching lights on and off and moving blinds (security); controlling of all the home elements (lighting, blinds, etc.) using a touch screen, a remote control, or an office computer (comfort and communication); and so on.

In order to realize all these tasks the HA system collects information from inputs or sensors (e.g., push buttons, temperature sensors, etc.), processes this information and activates outputs (lights, blind motors, etc.) using the appropriate actuators. Hence, HA implies the integration of different subjects such as electronics, computer science, and communication systems. Consequently, neither the development of HA applications nor its teaching is a trivial issue.

One of the main problems of HA development lies in the fact that there is no agreement in the standard to implement the applications. HA applications and devices currently belonging to different manufactures are isolated from each other thereby creating the main obstacle to HA market growth.

Leading companies in this market have adopted several standards and protocols. Some worth mentioning examples are the KNX (ISO/IEC14543-3 X), Lonworks (ISO/IEC 14908), and X10 (<http://www.x10.com>) technologies. Furthermore, as stated in Ref. 5, it is improbable that there will be a single dominant technology for HA in the short term. Each of these technologies provides its own software suite to create HA applications and program the devices. Hence the particular technology (specific platform) must be selected at the initial design stages, inasmuch as the tools and devices to be used depend on this choice. As V. Miori states, the cost of devices, the lack of user friendly interfaces and universal HA standard platforms (in which HA applications can interoperate) are factors which inhibit the spread of HA systems [5]. These issues have an important influence in the way that HA is

Correspondence to: M. Jiménez (manuel.jimenez@upct.es).

taught due to the fact that students learn using a specific platform from the first stages of development.

In addition, as in many other countries, the Spanish higher education system is currently being adapted to the Bologna process [6] which promotes changes in the educational model. As such, Universities are obliged to pay attention to these new challenges. In particular, teachers must provide their students with new skills, responsibilities, and knowledge. Consequently, teachers have the obligation to develop their educational programs and teaching techniques in order to fulfill all these issues. In this context, the time required to learn the concepts in each specific HA technology is important and the costs of providing laboratory equipment to train students in different technologies are substantial. Due to this, students neither have the time nor the resources to study a wide number of cases studies and lack the facilities to develop medium/large size HA applications. Thus, students invest too much time to learn specific HA execution platforms and as a consequence very few hours are made available to develop and deploy attractive HA examples. Therefore, it would be very useful to have a platform independent tool that allows students to implement HA applications using domain concepts and without being restricted by any specific technology. It is necessary to present educational materials to the students in both a useful and an appealing manner [3]. Accordingly the academic effort would be able to concentrate on learning the concepts and the logic necessary to implement the applications, thereby avoiding the complexities added by each specific technology.

This work proposes a platform independent tool which uses domain concepts to teach HA system development in higher education masters and degrees. It is an interesting academic tool like other academic tools developed to help students in different subjects like manufacturing systems [7], control engineering teaching [8], or welding processes [9].

The rest of this article is organized as follows: Second Section details the proposed learning tool for HA. Third Section describes the teaching methodology with the tool. Fourth Section develops an example of use of our tool for a particular application. Fifth Section summarizes the educational activities carried out with the tool. Sixth Section details the survey conducted amongst the students who used the tool, and finally seventh Section presents the conclusions and future work.

A PLATFORM-INDEPENDENT TOOL FOR HA TEACHING

As stated above, the aim of the developed tool is to provide users (students in our case) with an intuitive and flexible environment for developing HA applications. At the same time, this tool must be

independent of the final implementation platform. Thanks to this, students are given a better understanding of the concepts in the HA domain since these concepts are put in practice without the need to learn platform dependent (KNX, Lonworks, X10, etc.) issues. Furthermore, there is no need to use existing commercial tools for these technologies that are both difficult to use and have a high economical cost.

The developed tool incorporates a graphical language for modeling HA systems in a platform independent way (for details about the elements of the language see Ref. 10). Such language provides easy and intuitive descriptions of HA systems using graphic models. At the same time, the language is concise and translatable to any HA technology.

To define the language elements the following issues have been considered. Any HA system incorporates a number of *functional units* which are present in all the technologies and standards proper to the domain. These technologies differ in their architecture and the protocols they use, but they are identical when it comes to capability. In order to encourage the re use of these functional units and to avoid having to repeatedly define the same unit for each application, it was decided to use a catalog of reusable functional units (see Fig. 1). For their part, functional units have some *services* (reusable among the functional unit catalog) through which they are able to interact with other units.

Thanks to the catalog availability, any HA application is described by means of the instantiation of these functional units. These instances can be configured by adding the necessary values to their parameters. The links between functional unit services indicate the way in which these functional units will interact with the rest of the system.

Details of the Tool

The tool provides an integrated environment for teaching and learning HA design techniques without considering any specific platform. Figure 2 shows a screenshot of the tool where three parts can be distinguished:

- A drawing area to build graphic models for the applications. Models can be built either using the graphic palette or the pop up menus.
- A graphic palette that contains the elements which can be dragged to the drawing area (see Fig. 3).
- An area where the available properties (attributes, parameters, etc.) are displayed and can be modified for the selected element (functional unit or link). Properties are shown using editable text fields or drop down lists to provide an intuitive and easy access to them.

SENSOR DEVICES			ACTUATOR DEVICES			CONTROLLER DEVICES		
Symbol	Device	Category	Symbol	Device	Category	Symbol	Device	Category
WD Water Detector	Security		LO On/off Light	Energy / Comfort		SWO Switching Out	Switches
SD Smoke Detector	Security		LDM Dimmer Light	Energy / Comfort		SMS Mobile	Security
PIR3 Presence Detector	Security / Lighting		SHM Shutter	Energy / Comfort		MTO Motorization Out	Motors
SS Subset Sensor	Lighting / Motors		BL Alarm	Security		WT Week Timer	Lighting On / Off

Figure 1 Excerpt of the functional units catalog. [Color figure can be viewed in the online issue, which is available at www.intellect.com.]

Figure 2 Screenshot of the tool and its different subsections. [Color figure can be viewed in the online issue, which is available at wileyonlinelibrary.com.]

The model built with the graphical language is a directed graph where functional units are the vertices and links are the edges. On the left side input units like pushbuttons or sensors are located. On the right output elements such as lights or shutters are placed. Controllers are positioned in the middle, linked with the rest of the functional units.

For example, to instantiate the PB 1 functional unit (a push button) of Figure 2, first the *StandardFUnitInstance* element from the palette (see Fig. 3) must be dragged to the drawing area. Afterwards, this element must be selected to fill its properties in. This includes the selection of the particular functional unit from the catalog (see Fig. 1) in the *isInstanceOf* field, and the filling in of its name, references, and description (*FU Instance Name*, *FU Instance Ref*, and *FU Instance Des*) fields among others. Finally the functional unit parameters must be instantiated with

an actual value. For this, the *ParameterInstance* element with its corresponding value (String, Number, Boolean, Literal, etc.) must be dragged from the palette to the functional unit in the drawing area.

Implementation Technology

For the development of the tool described in this article the well known model driven development approach (MDD) [11] has been considered. In this approach, application code can be automatically generated from models at abstraction levels by means of transformations. The definition of the graphical language has been performed using domain specific language (DSL) techniques [12]. DSLs play an important role since they provide easy and intuitive descriptions of the system by means of graphic models.

Figure 3 Tool palette in detail: representation and description of some of the elements used in the graphical language. [Color figure can be viewed in the online issue, which is available at wileyonlinelibrary.com.]

The Eclipse framework [13] has been used for supporting our DSL implementation and model management. In particular, the Eclipse Modeling Framework (EMF) [14] plug in has been used to create the model editors and interoperate with other tools. For the DSL the Eclipse Graphical Modeling Framework (GMF; <http://www.eclipse.org/gmf>) was used, which helps to automatically generate graphic editors as Eclipse plug ins from models. Transformations from DSL models towards code generation have been performed with additional plug ins like EMT [15] and Java Emiter Templates (JET) [16].

We can find many examples in the literature of work performed using the Eclipse development environment applied to the field of education. The clearest are those applied in the teaching of software engineering, where students must learn to program in

different languages. Eclipse facilitates this task as it provides a series of plugins that provide a “perspective” on the environment for different languages. In addition, plugins have been developed to facilitate the learning of novice students. For example, PENUMBRA [17] or DrJava [18] plug in aim to make the learning process easier for students by removing certain features of the Eclipse environment.

Regarding the development of HA systems using an MDD approach the literature offers a few examples of works that try to reach it in an integrated way. Among these it is important to highlight the works of Munoz et al. [19], Voelter and Groher [20], and Nain et al. [21] that outline the necessity of using a model driven approach in HA systems development. These three works use Eclipse as development environment but none of them is focused on education.

Figure 4 Overview of the teaching methodology with the tool. [Color figure can be viewed in the online issue, which is available at wileyonlinelibrary.com.]

Figure 5 KNX scale model infrastructure. [Color figure can be viewed in the online issue, which is available at wileyonlinelibrary.com.]

OVERVIEW OF THE TEACHING METHODOLOGY WITH THE TOOL

The methodology here described considers two main roles: teacher and student. Taking into account the tasks performed by each role, the methodology can be deconstructed in the steps shown in Figure 4. The process starts when the teacher introduces, through several sessions, the theoretical concepts of the domain. Afterwards, the students are able to put into practice such learning. The teacher offers students a set of problems to be solved separately. Each problem includes a set of requirements to be validated using a HA scale model developed by us for educational purposes. These requirements will be related to security, comfort, energy saving, etc.

In the laboratory every student will be given the tool installed in their computer, a HA functional unit catalog, and the scale model infrastructure with platform specific devices (see Fig. 5).

The code needed to program the devices is automatically generated from the DSL model. Thus, students are able to validate the correct functioning of the system with respect to the initial set of requirements. The mentioned transformations (model to model and model to code) have been extended to support traceability. Traces allow students and teachers to check that all requirements have been taken into account and the targets (devices) where they have been deployed.

EXAMPLE OF USE

The aim of this section is to show to what extent the use of the tool can facilitate the development of medium to large size HA applications and the immediate increase in motivation and productivity in students. The following is a possible scenario proposed by the teacher to automate a home:

I want the hall light to switch on automatically when I go into the house. I want to have lights that I can regulate in my living room and bedrooms. In the balcony I'm concerned about security as it is easy to break in. At night I enjoy staying out so I would like lights to switch on automatically. I am also concerned about fire in the kitchen so if you could do anything about it that would be great. I want the shutters of the bedrooms to go up automatically in the morning and down at night. Last year I had a broken pipe at the bathroom and we had a flood. I want to have something that tells me if there is a flood. I also play poker every Friday at 9:00 pm so it would be great that the living room light turned on automatically.

The first step for the students would be to elicit the requirements of the application as shown in Table 1.

With these requirements the students build the DSL model. Figure 6 shows a four step process of modeling requirement

Table 1 List of Requirements for the Example of Use

Id	Category	Description
1	Comfort	Lights to switch on/off: hall, kitchen, balcony, and bathroom
2	Comfort	Lights to dim (regulate intensity): living room, bedrooms
3	Energy saving and comfort	Lights to switch with presence detection: hall
4	Comfort	Shutters to rise/lower with sunset sensor: bedrooms
5	Security	Water detection at bathroom
6	Energy saving	Light to switch with sunset sensor: balcony light
7	Comfort	Light on/off at the living room at 9:00 pm on Fridays with a weekly timer
8	Security	Smoke detector at the kitchen
9	Security and communications	Send SMS messages in case of flood or smoke detection
10	Security	Burglar alarm function with movement detector in the hall

Figure 6 Building the DSL model for requisite number 8. (a) Loading the functional unit catalog. (b) Instantiating functional units. (c) Linking functional units. (d) The requirement modeled with the tool. [Color figure can be viewed in the online issue, which is available at wileyonlinelibrary.com.]

number 8: the *StandardUnitInstance* from the tool palette (see Fig. 3) must be dragged to the drawing area. Afterwards, this graphic element must be edited selecting the *Smoke Detector* (SD) functional unit from the catalog of Figure 6a. The rest of the functional units are deployed following the same procedure.

The switching input controller SWI 1 is linked to the smoke detector in order to call the service *SWswitchOut* when a raising edge (*SWISendOnRisingEdge* parameter) is detected from SD 1. TM 1 implements a timer function to activate the bell BL 1 through the switching output controller (SWO 1).

Figure 7 DSL model created with the tool for the example of use. [Color figure can be viewed in the online issue, which is available at wileyonlinelibrary.com.]

Table 2 Curriculum of the HA Subject Without (Left) and With the Tool (Right)

Classroom lessons (15 h)		Classroom lessons (15 h)	
Introduction to HA	2 h	Introduction to HA	2 h
HA concepts and components	3 h	HA concepts and components	6 h
KNX technology	3 h	KNX technology	2 h
Lonworks technology	3 h	Lonworks technology	2 h
X10 technology	2 h	X10 technology	1 h
Other HA technologies	2 h	Other HA technologies	2 h
Laboratory (30 h)		Laboratory (30 h)	
KNX: Tool training with basic examples	6 h	Training in our tool and with basic examples	4 h
KNX: Small-size applications	8 h	Development of medium-to-large applications with our tool	16 h
Lonworks: Tool training with basic examples	6 h	KNX: Tool training with basic examples	4 h
Lonworks: Small-size applications	8 h	Lonworks: Tool training with basic examples	4 h
Other technologies (X10)	2 h	Other technologies (X10)	2 h

Figure 7 shows a screenshot of the tool with the DSL model resulted from integrating the previously described requirements. For example, requirements number 4 and 6 from Table 1 are located at the bottom of the drawing area. Here the sunset sensor SS 1 has a hardware channel connection to the SWI 5 input controller. This controller has the output service SWIswitchOut that is linked to the controllers SWO 3 and MTO 3. SWO 3 is a functional unit which turns on and off the balcony light (LO 2). MTO 3 is an output controller to move the bedroom blind SHM 1. As can be seen for some requirements, integration of different devices is a key point of the HA domain. For example, the movement detector PIR2 1 is used both to switch the hall light (LO 1) on/off and to fire the burglar alarm when requirements numbers 3 and 10 are implemented.

A picture of our scale model for the KNX technology is shown in Figure 5. This infrastructure has been built to enable the students to validate models created with the DSL.

EDUCATIONAL ACTIVITIES CARRIED OUT WITH THE TOOL

During the past academic course 2008/2009, the decision was taken to use the previously described tool in the educational activities of the HA subject of the Master in Industrial Informatics, given in the Technical University of Cartagena. Table 2 shows the curriculum and the time distribution for the HA subject before and after introducing the tool presented in this article.

The previous curriculum did not allow the development of complex applications since time was invested in creating small size platform specific applications. Theoretical classes were reorganized thereby giving more time to learn HA domain concepts and components rather than to study particular technologies. Note the reduction of the specific platform laboratory work which led to an important increase in the time available for developing more complex applications with our tool (highlighted in Table 2). Furthermore, availability of this tool for the students promotes autonomous study since commercial tools for specific platforms are subject to licenses so they can only be used by students in the laboratory computers.

EVALUATION OF THE TOOL

It is important for software products to be evaluated for each relevant quality factor using widely accepted metrics [22]. In order to evaluate the tool and the benefits offered by the use of the DSL we conducted an experiment involving a group of 42 students on

the Master in Industrial Informatics during the academic course 2009/2010. The students were enrolled in the subject of HA and the experiment was mainly concerned with the *usability* feature: capacity of a software to be understood, learnt, used and be attractive for the user, under specific conditions [23]. The experiment took into consideration the following usability quality factors: understanding, ease of learning, operability, flexibility, accordance, and attractiveness. We presented participants with a case study in which they were to use both tools (these students had already attended an extra training course for the ETS tool). Afterwards, they were asked to complete a questionnaire rating their experience (using a 1 to 5 point scale, where 5 is the highest quality rating). We obtained final valuations for each factor using an arithmetic mean of the results of each question. Figure 8 shows the results of the survey.

Study participants rated the DSL tool higher than the commercial tool (ETS). The largest differences were in understanding, ease of learning, and attractiveness. For example, to questions such as, “Do you need help to remember the concepts represented by each primitive in the palette/tool bar?” most students answered “not at all” (score 5) in our tool questionnaire. Fewer students did so for the commercial tool. Differences in factors such as flexibility and operativity are lower.

CONCLUSIONS

In this article a new tool to teach HA systems development is presented. The developed tool is chosen as a good alternative on the basis of its expected reduction in the teaching hours for HA applications. This software shows HA specific properties and concepts and allows students to model and implement medium

Figure 8 Results of the usability factors survey. [Color figure can be viewed in the online issue, which is available at wileyonlinelibrary.com.]

to large size applications within a few minutes. When compared to other HA based tools, the main differences are the use of an extensible catalogue of platform independent functional units and a DSL for modeling. Because of this, the tool has been adopted as support for an accompaniment to theoretical lessons in the subject HA of the Industrial Technologies Master Degree at the Technical University of Cartagena. A video demonstration of the tool can be accessed through this link: <http://repositorio.bib.upct.es/dspace/bitstream/10317/854/6/has.swf>.

Students have benefited from the use of the tool because they can easily obtain and model HA applications in a considerable shorter time. Furthermore, as the tool is handed to the students they are able to develop several examples as part of their home work (as Bologna promotes the development of self study and work at home). This would not be possible with commercial tools like ETS (Engineering Tool Software for KNX) or Lonmaker (the development tool for Lonworks) that require purchasing the respective licenses leading to high costs. This new method meant that students have been more motivated and have scored higher marks. These results could be linked to the fact that the tool allows students to get a deeper understanding of HA concepts and not just programming and platform dependent matters.

An experiment to evaluate the usability of our tool was performed, where it was compared with a commercial tool. As a result, our tool was rated higher than the commercial one. Additional experiments are planned as future work in order to validate the completeness of the tool and analyze factors such as development time. All this information will be very valuable to extend and tune future versions of the tool to student's needs.

We are now focused on defining the transformation mechanisms for other execution platforms and the interoperability of our tool with software to validate the models before final deployment. In addition, work is currently underway to improve the traceability tool and, lastly, the integration of the tool in environments other than Eclipse.

REFERENCES

- [1] M. Chana, E. Campoa, D. Estèvea, and J. Y. Fourniols, Smart homes: Current features and future perspectives, *Elsevier Maturitas J* 64 (2009), 90–97.
- [2] L. Jiang, D. Y. Liu, and B. Yang, Smart home research, machine learning and cybernetics, *Proceedings of 2004 International Conference, Vol. 2*, 2004, pp 659–663.
- [3] L. Burnell, A. Sanchez, J. Priest, and C. Hannon, The Crescent Lab: A smart home lab for students, *Computer Science, ENC '06. Seventh Mexican International Conference*, 2006, pp 55–61.
- [4] C. R. Baker, Y. Markovsky, J. van Greunen, J. Rabaey, J. Wawrzynnek, and A. Wolisz, ZUMA: A platform for smart-home environments the case for infrastructure van Intelligent Environments, *IE06, 2nd IET International Conference*, Vol. 1, 2006, pp 257–266.
- [5] V. Miori, L. Tarrini, M. Manca, and G. Tolomei, An open standard solution for domotic interoperability, *IEEE Trans Consum Electron* 52 (2006), 97–103.
- [6] European Ministers of Education. The Bologna Declaration of 19 June 1999, <http://www.bologna-berlin2003.de/pdf/bologna-declaration.pdf>.
- [7] M. A. Martínez and M. Cañadas, A tool for the educational study of manufacturing systems, *Comput Appl Eng Educ* 18 (2007), 130–143.
- [8] J. A. Méndez, C. Lorenzo, L. Acosta, S. Torres, and E. González, A Web-based tool for control engineering teaching, *Comput Appl Eng Educ* 14 (2006), 178–187.
- [9] R. H. Gonçalves, J. C. Dutra, and G. Locatelli, A didactic computational tool for monitoring and control of arc welding processes-teaching and research, *Comput Appl Eng Educ* (2009), Published online in Wiley Online Library; DOI: 10.1002/cae.20390.
- [10] M. Jimenez, F. Rosique, P. Sánchez, B. Álvarez, and A. Iborra, Habitation: A domain-specific language for home automation, *IEEE Software*, 26 (2009), 33–38.
- [11] B. Selic, The pragmatics of model-driven development, *IEEE Software*, 20 (2003), 19–25.
- [12] M. Memik, J. Heering, and A. M. Sloane, When and how to develop domain-specific languages, *ACM Comput Surv* 37 (2005), 316–344.
- [13] Eclipse.org home, <http://www.eclipse.org>.
- [14] Eclipse Modeling Framework Project, <http://www.eclipse.org/modeling/emf>.
- [15] E. Biermann, K. Ehrig, C. ErmeI, C. Köhler, G. Kuhns, and G. Taentzer, Tiger EMF model transformation framework (EMT), <http://tfs.cs.tu-berlin.de/emftrans/papers/userdoc.pdf>.
- [16] Eclipse Consortium, Java Emitter Templates (JET), <http://www.eclipse.org/modeling/m2g/?project=jet>.
- [17] F. Mueller and A. L. Hosking, Penumbra: An Eclipse plugin for introductory programming, *Proceedings of the 2003 OOPSLA Workshop on Eclipse Technology eXchange*, 2003, pp 65–68.
- [18] C. Reis and R. Cartwright, A friendly face for eclipse, *Proceedings of the 2003 OOPSLA Workshop on Eclipse Technology eXchange*, 2003, pp 25–29.
- [19] J. Muñoz, V. Pelechano, and C. Cetina, Implementing a pervasive meetings room: A model driven approach, *Proceeding of the 3rd International Workshop on Ubiquitous Computing*, 2006, pp 13–20.
- [20] M. Voelter and I. Groher, Product line implementation using aspect-oriented and model-driven software development, *Proceedings of the 11th International Software Product Line Conference*, 2007, pp 233–242.
- [21] G. Nain, E. Dauber, O. Barais, and J. M. Jézéquel, Using mde to build aschizofrenic middleware for home/building automation, In *Service-Wave'08: Networked European Software & Services Initiative (NESSI) Conference*, 2008.
- [22] A. Seffah, M. Donyae, R. B. Kline, and H. K. Padda, Usability measurement and metrics: A consolidated model, *Softw Qual J* 14 (2006), 159–178.
- [23] J. P. Tolvanen and M. Rossi, MetaEdit+, defining and using domain-specific modeling languages and code generators, *Proceedings of the 18th Annual ACM SIGPLAN Conference in Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA 03)*, ACM Press, 2003, pp 92–93.

BIOGRAPHIES

Manuel Jiménez is an associate professor of industrial electronics at the Technical University of Cartagena and is a member of the university's DSIE (Division of Systems and Electronic Engineering) research group. His research interests include electronics and model driven engineering for reactive systems. Jiménez has a PhD in computer science from the Technical University of Cartagena.

Bárbara Álvarez is a full professor in computer science at the Technical University of Cartagena and a member of the university's DSIE (Division of Systems and Electronic Engineering) research group. Her research interests include real-time systems and software architectures for teleoperation. Alvarez has a PhD in telecommunication engineering from the Technical University of Madrid.

Pedro Sánchez is an associate professor of computer science at the Technical University of Cartagena and a member of the university's DSIE (Division of Systems and Electronic Engineering) research group. His research interests include model-driven engineering for real-time systems. Sánchez has a PhD in computer science from the Technical University of Valencia.

Andrés Iborra is full professor and head of the Electronics Technology Department at the Technical University of Cartagena and a member of the university's DSIE (Division of Systems and Electronic Engineering) research group. His research interests include computer vision and robotics. Iborra has a PhD in industrial engineering from the Technical University of Madrid.

Francisca Rosique is an assistant professor and a PhD student in computer science at the Technical University of Cartagena and a member of the university's DSIE (Division of Systems and Electronic Engineering) research group. Her research interests include model-driven engineering and home automation systems. Rosique has a master's in telecommunication engineering from the Technical University of Cartagena.

Impact Factor Trend Graph: COMPUTER APPLICATIONS IN ENGINEERING EDUCATION

Click on the "Return to Journal" button to view the full journal information.

**Impact Factor -- see below for calculations*

The journal impact factor is a measure of the frequency with which the "average article" in a journal has been cited in a particular year. The impact factor will help you evaluate a journal's relative importance, especially when you compare it to others in the same field. For more bibliometric data and information on this and other journal titles click on the "Return to Journal" button.

NOTE: Title changes and coverage changes may result in no impact factor for one or more years in the above graph.

2010 Impact Factor

Cites in 2010 to articles published in:	2009 = 21	Number of articles published in:	2009 = 47
	2008 = 6		2008 = 37
	Sum: 27		Sum: 84

Calculation: $\frac{\text{Cites to recent articles}}{\text{Number of recent articles}} = \frac{27}{84} = \mathbf{0.321}$

2009 Impact Factor

Cites in 2009 to articles published in:	2008 = 3	Number of articles published in:	2008 = 37
	2007 = 10		2007 = 27
	Sum: 13		Sum: 64

Calculation: $\frac{\text{Cites to recent articles}}{\text{Number of recent articles}} = \frac{13}{64} = \mathbf{0.203}$

2008 Impact Factor

Cites in 2008 to articles published in:	2007 = 8	Number of articles published in:	2007 = 27
	2006 = 18		2006 = 40

Sum: 26
Calculation: $\frac{\text{Cites to recent articles}}{\text{Number of recent articles}} = \frac{26}{67} = \mathbf{0.388}$

Sum: 67

2007 Impact Factor

Cites in 2007 to articles published in: 2006 = 14 Number of articles published in: 2006 = 40
2005 = 8 2005 = 31
Sum: 22 Sum: 71

Calculation: $\frac{\text{Cites to recent articles}}{\text{Number of recent articles}} = \frac{22}{71} = \mathbf{0.310}$

2006 Impact Factor

Cites in 2006 to articles published in: 2005 = 3 Number of articles published in: 2005 = 31
2004 = 9 2004 = 24
Sum: 12 Sum: 55

Calculation: $\frac{\text{Cites to recent articles}}{\text{Number of recent articles}} = \frac{12}{55} = \mathbf{0.218}$

[Acceptable Use Policy](#)
Copyright © 2012 [Thomson Reuters](#).

THOMSON REUTERS

Published by Thomson Reuters

Rank in Category: **COMPUTER APPLICATIONS IN ENGINEERING EDUCATION**

Journal Ranking

For **2010**, the journal **COMPUTER APPLICATIONS IN ENGINEERING EDUCATION** has an Impact Factor of **0.321**.

This table shows the ranking of this journal in its subject categories based on Impact Factor.

Category Name	Total Journals in Category	Journal Rank in Category	Quartile in Category
COMPUTER SCIENCE, INTERDISCIPLINARY APPLICATIONS	97	95	Q4
EDUCATION, SCIENTIFIC DISCIPLINES	33	28	Q4
ENGINEERING, MULTIDISCIPLINARY	87	62	Q3

Category Box Plot

For **2010**, the journal **COMPUTER APPLICATIONS IN ENGINEERING EDUCATION** has an Impact Factor of **0.321**.

This is a box plot of the subject category or categories to which the journal has been assigned. It provides information about the distribution of journals based on Impact Factor values. It shows median, 25th and 75th percentiles, and the extreme values of the distribution.

Key

COMPUTER
A - SCIENCE,

INTERDISCIPLINARY
APPLICATIONS
EDUCATION,
B - SCIENTIFIC
DISCIPLINES
ENGINEERING,
C - MULTIDISCIPLINARY

[Acceptable Use Policy](#)
Copyright © 2012 [Thomson Reuters](#).

THOMSON REUTERS

Published by Thomson Reuters

Introducing Safety Requirements Traceability Support in Model-Driven Development of Robotic Applications

Pedro Sánchez, Diego Alonso, Francisca Rosique,
Bárbara Álvarez, *Member, IEEE*, and Juan A. Pastor

Abstract Requirements traceability is a technique intended to help determine the impact of changes in software design, support their integration, preserve knowledge, and assure the quality and correctness of the overall system. This paper presents an approach that considers traceability of safety requirements in the context of model-driven development of teleoperated service robots. The combination of the model-driven approach with safety requirements traceability makes it possible to construct systems using techniques for automatically identifying, managing, and mitigating risks so that these systems are safe enough to work in a particular environment. To secure the advantages of these mechanisms, we have developed a tool that provides users with traceability reports after applying model transformations. These reports enable developers to determine whether or not all safety requirements have been considered, the impact of changing a safety requirement, and how they are considered both in architectural decisions and code implementations.

Index Terms Model-driven engineering, traceability, safety systems, service robots.

1 INTRODUCTION AND MOTIVATION

HUMAN operators use teleoperated service robots to perform hazardous operations in hostile environments, such as nuclear reactors, space missions, and warehouses. Since service robots have to interact with both the environment and human operators, it is essential that they be so designed as to involve no risk to the operators, the environment, or the robot itself. Where it is impossible to eliminate the risk, this at least must be limited to an acceptable level. Safety requirements should be identified in the same way as any other system requirements, but this requires more detailed study in view of their importance. Also, safety has a major impact on both the specification and the design phases, especially when designing the system architecture.

Robotic systems comprise numerous embedded devices that should work cooperatively in a safe manner. Embedded system programming implies the use of schedulability theories and formal analysis techniques to ensure timing requirements and correctness. It also implies the adoption of design patterns aimed to achieve the reliability and fault-tolerance levels that ensure a safe operation. In general, the design of a robotic system should integrate knowledge and tools of different engineering domains.

With more than 12 years of experience in the design of software applications for teleoperated service robots [1], the

DSIE research group at the Technical University of Cartagena has developed several teleoperated service robots in the context of the European Union V Framework Programme *Environmental Friendly and Cost-Effective Technology for Coating Removal* (EFTCoR) project [2]. The EFTCoR project addressed the development of a solution to the problem of retrieval and confinement of subproducts from ship-maintenance operations. In such a dangerous and hazardous environment, it was essential to address both the hardware and software safety requirements from the outset of the project in order to design safe robots. For this purpose, we used the ANSI/RIA 15.06-1999 standard [3] as the basis for identification and classification of the hazards and risks and the safeguards to be adopted in order to reduce the risks to acceptable levels. This standard was completed with the contributions of *Rapid Object-Oriented Process for Embedded Systems* (ROPES) [4] for dealing with complex software-based solutions, since ROPES provides some guidelines for gathering and classifying safety-related requirements. A detailed study of the results of this combination in the context of the EFTCoR project can be found in [5].

Also, the DSIE group recently achieved good results [1] by adopting the *Model Driven Engineering* (MDE) approach [6] for the development of service robots. MDE proposes the use of models as the principal artifact for software development. A model is a simplified depiction of reality, which shows only the aspects that are of interest for a given purpose and leaves out those that are not. In this approach, a model is defined in accordance with a metamodel, which defines the abstract syntax of a modeling language and establishes the concepts and the relationships between them, also including the rules that determine when a model is properly formed. Any given system can be described by different models at

- The authors are with the Division of Systems and Electronic Engineering (DSIE), Technical University of Cartagena, Campus Muralla del Mar s/n, E 30202 Cartagena, Spain.
E mail: {pedro.sanchez, diego.alonso, juanangel.pastor, balvarez, francisca.rosique}@upct.es.

Manuscript received 26 June 2009; revised 19 Nov. 2010; accepted 18 Mar. 2010; published online 11 June 2010.

Recommended for acceptance by S.K. Shukla and J. P. Talpin.

For information on obtaining reprints of this paper, please send e mail to: tc@computer.org, and reference IEEECS Log Number TCSI 2009 06 0305.
Digital Object Identifier no. 10.1109/TC.2010.149.

different levels of abstraction, and so model transformations [7] (*model-to-model (M2M)* and *model-to-text (M2T)*) are one of the key issues of this approach.

In a previous work [8], the DSIE group designed a component metamodel (V^3CMM) and a development process for modeling the software architecture of the robots it designed. V^3CMM provides the facilities for platform-independent modeling of the components that comprise the architecture of the application and of their behavior, while a set of model transformations enable code generation, taking into account different target implementation platforms (e.g., object-oriented languages and component frameworks) and analysis tools.

The development of safety systems may also benefit considerably from the combination of MDE and traceability-related techniques. The term “traceability” is defined in the IEEE Standard Glossary of Software Engineering Terminology [9] as: “*the degree to which a relationship can be established between two or more products of the development process, especially products having a predecessor-successor or master-subordinate relationship to one another; for example, the degree to which the requirements and design of a given software component match.*” A MDE development environment can greatly facilitate traceability by making it possible to record links between source and target elements while generating other software artifacts by means of model transformations. These links can be useful for performing impact analysis, model-model synchronization, and model-based debugging and determining the source or target of a transformation. Links make it easier to validate whether transformations are properly carried out and whether the artifacts produced in the course of the transformation process correctly satisfy the stakeholder requirements. In this way, developers can validate the completeness and correctness of transformations, obtain support for integrated management of changes in system requirements, and assess the impact of the changes on the different levels of abstraction (requirements, architectural design, detailed design, and lastly executable code).

MDE can provide a suitable theoretical and technological support for integrating different facets of safety-critical system in a global development framework. To implement a system that assures safety, MDE, and traceability, there are a number of conceptual and development (technical, tools, etc.) requirements that have to be met. The absolutely essential ones are basically:

1. An integrated framework of safety management requirements.
2. A catalog of architecture solutions for these requirements.
3. A framework for automatic generation of code for different implementation platforms.
4. Support for traceability of the various artifacts involved in the development.
5. Tools which enable fully automated handling of all the information involved in the process.

This paper presents an integrated environment that furnishes these resources and demonstrates the viability of the approach on the basis of experience acquired in the development of an industrial robotic system with safety requirements.

This paper describes a relevant case study in synergy between the Robotics and Software Engineering domains.

While MDE is a well-known Software Engineering paradigm and Robotics is a discipline with a long tradition, as far as we know, there is no experience on Robotics that uses in an extensive way the MDE approach. Nevertheless, there are tremendous opportunities to start with reusable and semantically well-defined designs of complex robotic software systems [10]. The work described in this paper offers developers an MDE environment which: 1) promotes the reuse of the software artifacts generated in the safety systems construction process and 2) considers the traceability of these artifacts in order to enhance the quality of the systems, since it facilitates both modification and analysis of the impact of the changes on the safety requirements. It is worth noting that these results can be easily extrapolated to other domains different to robotics, since the software artifacts developed and used are indeed domain independent.

The rest of this paper is organized as follows: Section 2 gives an overview of the safety standards and basic concepts that are used throughout the paper. Section 3 offers an overview of the previous works of the DSIE that motivated this one. Section 4 details the proposed framework for integrated management of safety requirements. Section 5 describes the elements that are indispensable to the implementation of a traceability approach within a MDE framework. Section 6 analyzes related work, and finally, conclusions and further work are presented in Section 7.

2 SAFETY STANDARDS AND BASIC CONCEPTS

There are several approaches to manage safety in the literature (see [11] for details about the most commonly used hazard-analysis methodologies in the system safety discipline), but they are either too general or not specifically targeted to the teleoperated robot domain. Some try to define a standard that guides the whole process (from requirement identification to the final solution), while others simply provide tools or techniques. Among these standards, it is worth highlighting the European Standard EN 61508:2001 [12] as well as the already mentioned American ANSI/RIA R15.06-1999 [3].

The four-step methodology presented in [5] proposes the fusion of these standards. It encourages the tracking of safety throughout the complete life cycle of the robot (as EN 61508 proposes); also, it uses the ANSI/RIA standard as a guide to classify hazards and to propose solutions, completing them with the contributions of ROPES [4] for dealing with complex software-based solutions. The different steps that make the methodology up are summarized as follows:

STEP 1: Identify hazards. Hazard analysis includes identification of the possible causes of the failure (hardware, software, or human), the tasks in which it can happen, and the reaction to the occurrence of the hazard.

STEP 2: Identify risks. Identify the possible risks of the system, classify them according to the impact they have on the system and the environment, and link them to the hazards identified in the first step. The ANSI/RIA standard characterizes risk by its *Severity*, *Exposure*, and *Avoidance*. Each of these has two different values (high/low), giving eight possible combinations. Depending on the values of

Fig. 1. Crane robot for cleaning vertical surfaces in EFTCoR. Two images on the left: primary positioning system; two images on the right: detail of the XYZ table (secondary positioning system) and the blasting tool.

these characteristics, a *Risk Reduction Category (RRC)* is identified. Based on the RRC, ANSI/RIA demands a certain level of performance of the safeguards and circuits that are to be designed to reduce the risk (Simple, Single Channel, and Single Channel with Monitoring and Control Reliable). Moreover, ANSI/RIA also recommends the adoption of safety policies (like adequate training, presence detectors, security barriers, etc.) to help human operators avoid some risks. After applying the safeguards designed for the specific RRC, a new analysis is performed to calculate the residual risk, just to be sure that the risk is kept at a tolerable level for both the system and the environment. This has to be repeated during the life cycle of the system to ensure that no new risk appears, and that the risks already identified are kept under control.

STEP 3: Specify safety requirements. Extract the system safety requirements from the results of the previous steps. This step is quite difficult to perform because neither the ANSI/RIA standard nor ROPES offer a methodology to deduce the requirements from the previous results; therefore, extraction is guided by the experience of the developers. To tackle this third step, it is necessary to have or develop: 1) an appropriate process for requirement harvesting, 2) a way to catalog them for reuse in other applications, and 3) tools to trace the use of the requirements throughout the development process.

STEP 4: Make safe designs. The design of the system's software architecture must consider the safety requirements, avoid failures that spread through the whole system, and be periodically reviewed when new hazards are identified. This objective requires a rigorous architectural approach that allows the architecture model to evolve when new requirements or conditions arise. The architecture design benefits greatly from the application of design patterns to resolve typical safety requirements [13], [14].

As discussed in Sections 4 and 5, this paper completes the authors' previous contributions with specific solutions to the last two steps in the methodology described.

3 PREVIOUS WORK OF THE DSIE GROUP

In order to facilitate understanding of the results reported in this paper, following is a summary of the most relevant research previously conducted by the DSIE Group. This section presents two perspectives of interest: our experience in the development of safe teleoperated service robots, and the adoption of MDE as a support both for reuse and for automatic code generation. These two prior pieces of research constitute the foundation on which the safety requirements traceability work reported here rests.

3.1 The EFTCoR System

The EFTCoR family of robots [1], [2] offers a global environmental-friendly solution to the problems related to hull maintenance operations. These operations consist of the removal of sea adherences and the hull coating followed by hull repainting. The solution developed in the context of the EU EFTCoR project comprised two families of robots:

- Teleoperated cranes with blasting tools for cleaning big areas, normally a ship vertical surface.
- Teleoperated climbing vehicles for cleaning different areas.

Cranes (see Fig. 1) consist of a primary positioning system, capable of covering large hull areas, and a secondary positioning system, mounted on the primary system, that can position a tool over a relatively small area (from 1 to 16 m²). These robots have been developed to perform the current hull cleaning operations in a way that avoids residue emissions to the environment and enhances the working conditions of the shipyard operators. The fundamental characteristics of the EFTCoR that compel a safety-driven approach when designing the robots are the following:

- The operator uses a heavy mechatronic device whose range of movement can cause serious damage.
- Blasting of the hull with high-pressure abrasive particles. The power of the jet makes it dangerous for human operators and for equipment. In addition, the

TABLE 1
Excerpt of Tasks

Task	Type	Description
T1	operator	Move the primary positioning system (rail).
T2	operator	Move the primary positioning system (vertical axis).
T3	operator	Move the primary positioning system (arm).
T4	operator	Coordinate the primary positioning systems for positioning the secondary positioning system (XYZ table).
T8	operator	Execute a sequence of movements of the primary positioning system.
T12	operator	Stop the primary positioning system (rail).
T13	operator	Stop the primary positioning system (arm and vertical axis).
T20	maintenance	Calibrate primary positioning system.

impact of the jet on the ship hull produces dust, aggravating workplace conditions.

- The system has been designed for working outdoors. There is a lot of movement in the robot's working environment (shipyards): many cranes handling heavy burdens, operators moving around, etc.

The EFTCoR crane is 12 m high and weighs 20 tons, which means that robot movement must necessarily take safety requirements into consideration. Also, in its central zone, it has a 2-ton jointed arm for holding the XYZ table (which includes a cleaning tool). The control system for the XYZ table has been designed to follow cleaning instructions from a human operator or from a computer vision system,

which locates the areas of the hull that have to be blasted and commands the crane robot.

Tables 1, 2, and 3 contain extracts from the results produced by the methodology described earlier for identifying and analyzing safety requirements for the EFTCoR crane robot. It is important to note that the solutions listed are very high level and are reported from the standpoint of the problem domain (see Table 3, "Safety Requirement" column). For more detailed information on the safety study conducted for the EFTCoR system, see [5].

Following is a description of the architecture solution that was adopted to deal with the H13 threat. We shall return to this example later on to demonstrate the use of the tools and techniques presented in this paper. A detailed analysis of hazard H13 (see Tables 2 and 3) suggests the following possible sources of error: 1) failure of any of the sensors integrated with the motors that move the arm, 2) the electric power is off, and 3) the control unit is not working properly (hardware failure or software error). The solution to hazard H13 is based on the well-known watchdog pattern [14]. Fig. 2 shows the architecture design with a component diagram that accomplishes the required risk reduction for H13. The WatchdogUnit node in the figure is the redundant component according to the watchdog pattern.

In order to help fully understand the traceability relationships defined in the next part of the paper, we have labeled all the elements used in the design of the WatchdogUnit component, including interfaces (offered and required) and ports.

3.2 V³ Component Metamodel

The paper [1] by Iborra et al. reports our experience in dealing with highly variable requirements of service

TABLE 2
Excerpt of Hazards

Hazard	Tasks/Risk Level/Prob.	Origins (faults)	Reaction measures
H4. Equipment in the trajectory of the primary positioning system (rail).	T1, T20 / Severe / Med.	There is an obstacle in the trajectory of the primary positioning system.	Emergency alarm. Stop the robot. Emergency stop.
H7. Primary pos. sys. limit sensor overcome (arm).	T3-4, T20 / Very severe / Low	Sensor breakage or software error. Comm. failure.	Emergency alarm. Emergency stop.
H13. The primary pos. sys. does not stop.	T1-4, T8, T12, T13, T20 / Very severe / Low	Error in the primary positioning system controller. Comm. failure. Power failure.	Emergency alarm. Power robot off. Emergency stop.

TABLE 3
Excerpt from Hazard List with Adopted Solutions, Indicating the SEAR (Severity, Exposure, Avoidance, and RRC) Level According to ANSI/RIA Standard, before and after the Adoption of the Indicated Solution

Hazard	Risk	SEAR pre	Safety Requirement (hazard solution)	SEAR post
H4. Equipment in the trajectory of the primary positioning system (rail).	R4. Damage to the equipment and to the primary positioning system.	High S. High E. Low A. R2A	Add sensors to detect obstacle presence in the rail. These sensors are directly wired to the robot safety control infrastructure. Add a siren for signalling robot motion.	High S. Low E. Low A. R3B
H7. Primary pos. sys. limit sensor overcome (arm).	R7. Damage to the primary pos. sys. or workers.	High S. Low E. High A. R2B	Add mechanical limits	Low S. Low E. Low A. R4
H13. The primary pos. sys. does not stop.	R8. Damage to the primary positioning system or workers or equipment.	High S. Low E. High A. R2B	Add additional emergency stop mechanisms. Add external motion sensors.	Low S. Low E. Low A. R4

Fig. 2. Component diagram for solving the hazard H13.

robotics applications ranging from teleoperated systems for specific tasks in fully structured environments to semiautonomous robots in partially structured environments. It documents how different Software Engineering techniques have been progressively exploited for the development of reusable control systems and how the software reuse practice has been enhanced from pure code to abstract models, i.e., from implementation to design. The challenge was to find an approach, whereby the code could be reused in applications with different architectures. The solution arrived at was to adopt the *Component-Based Development* paradigm (CBD) [15]. CBD is conceived for the purpose of speeding up the software development process. It states that such development would be achieved by linking independent parts, the components. A software component is a unit of composition with contractually specified interfaces and explicit context dependencies only, which can be deployed independently.

Then there was a need for an approach that came with a set of tools to facilitate automatic or semiautomatic generation of applications. These needs have been satisfied by adopting the well-known MDE approach, resulting in the development of an integrated reactive system development environment designed by the DSIE research group. It offers the following features:

1. It considers the component metamodel called V³CMM, which is an expressive and easy to use metamodel, designed in order to develop general purpose component-based applications (both their structure and the behavior of their components) in a platform-independent way and to maximize model reuse.
2. It relies on model transformations as the fundamental tool for automatic code generation. A M2M transformation has been created to translate the CBD concepts appearing in V³CMM (e.g., port, component, etc.) into object-oriented concepts (e.g., class, operation, etc.). In this case, UML has been chosen as

the target metamodel to express this object-oriented model, as it provides the required modeling capabilities and low-level primitives. This intermediate UML model is mainly meant to facilitate code generation for different object-oriented languages, and to allow integration with other tools (such as UML verifiers, animators, etc.). V³CMM adopts the UML definition of the concept “component,” and the transformation aims to generate an evolvable object design that fulfils the main properties of the chosen definition, namely that components shall have runtime independence, that they shall encapsulate its inner structure, and that components shall issue operation calls exclusively through their ports. Several design patterns [13] have been applied in order to achieve these properties, such as Active Object, Proxy, Façade, Command, and State, to mention a few.

3. Finally, the process is completed by the definition of a M2T transformation from the object-oriented UML model to the Ada 2005 programming language, which generates the Ada skeleton for the V³CMM model. This process is flexible enough to define additional M2T transformations for generation of code for different programming languages. Also, additional transformations may be defined to create suitable models for analyzing the properties of the V³CMM model, as there are a number of tools in the marketplace that perform different types of analysis.

All these resources have been integrated in the Eclipse development environment using the following MDE-related plug-ins: Eclipse Modeling Framework (EMF) [16], EMF Object Constraint Language (EMF OCL), which provides a formal language for defining constraints and queries on models, Atlas Transformation Language (ATL) [17], which adds a declarative M2M transformation language, and Java Emitter Templates (JET) [16], which adds a template-based M2T transformation language).

Fig. 3 shows part of the UML model generated automatically from the component diagram in Fig. 2 after applying the M2M transformation from the V³CMM component model to the UML object model. Only the UML elements (classes, interfaces, methods, and attributes) relating to the WatchdogUnit are shown. This UML model is used in Section 5 to illustrate the use of the traceability tool that has been developed.

4 A FRAMEWORK FOR SAFETY REQUIREMENTS MANAGEMENT

Fig. 4 shows a general schematic of the artifacts and tools used in the proposed process. There are two clearly differentiated parts: 1) the part relating to management of safety requirements (“Traceability Management”) and 2) the part integrated in the application software modeling process and code generation (“V³CMM Framework”).

As noted earlier, management of safety requirements from a MDE perspective requires the design of a metamodel to represent them. Every system that is built will have a defined set of safety requirements, possibly reused ones from similar experiments or even from outside catalogs.

Fig. 3. Excerpt from the UML model generated after execution of the M2M transformation, showing only the class diagram for the WatchdogUnit component.

These are the requirements in the “Safety Requirement Model,” which according to MDE nomenclature conforms to the metamodel described at the end of this section.

The process works as follows: The user starts modeling the safety requirements or reusing requirements that have been defined previously. The user must then develop an architecture solution (a V³CMM model) that meets the safety requirements, using known patterns [14] for the

architecture design of safety systems. The traceability matches between safety requirements and the elements of architecture design that satisfy them must be noted in the model called “Traceability Model #1,” following the metamodel described in Section 5.1. These annotations are created manually and are used as inputs for the tool described in Section 5.2.

The possibility of automatically deriving the architecture of the application from the chosen safety requirements is very desirable. However, this feature is constrained to those domains in which there is a previous accepted design, for instance, product lines or domains with very stable and known features (such as avionics). We consider that in most of systems, it is very difficult to find these conditions, in particular, in service robots, there is neither common architecture nor stable requirements shared by many systems. A solution to this problem requires the definition of domain-analysis technique extensions, together with the use of quality attribute templates [18].

Different requirement metamodels can currently be found in the literature [19], each one including different concepts and relationships. Such metamodels improve requirement reuse and serve as a structured requirement reference model. However, these are too general, and therefore, we preferred to define a metamodel to explicitly describe the safety concepts dealt with in the ANSI/RIA standard (hazards, risk, and risk reduction category, etc.). This solution is not incompatible with the adoption of a generic requirement metamodel (like the one described in [20], for example). Indeed, a generic metamodel is particularly useful for modeling the rest of the system requirements and does not prevent their integration with the metamodel concerned with safety. Fig. 5 shows the

Fig. 4. General MDE development framework for component-based applications, completed with safety requirements traceability management.

Fig. 5. Metamodel for safety analysis according to ANSI/RIA. The gray classes represent the safety requirements for a particular system, while the other classes show the values predefined according to that standard.

metamodel defined to represent the safety requirements, called the “Safety Requirement Metamodel.”

The information presented in Tables 1, 2, and 3 is recorded in the development tool using that metamodel. The TaskType class models the types of tasks that normally arise in the development of teleoperated robotic systems (see the examples in Table 1). The RiskLevel, SeverityLevel, RRC, ExposureLevel, AvoidanceLevel, and Probability classes contain the values predefined by the ANSI standard (see the values cited in Tables 2 and 3). The Hazard, Risk, and HazardSolution classes contain the hazards, risks, and solutions commonly adopted in systems of this type. In this context, these data constitute a reusable catalog of safety requirements in much the same way as with other requirements (for an example applied to security requirements, see [21]).

The set of safety requirements is only a part of the system requirements. These requirements complement others, such as requirements concerning functionality, performance, usability, and portability. Fig. 6 shows a complete schematic of this integration. As we can see, once all the recommendations and techniques for identification of safety requirements have been applied, the result is a document called “Safety requirements for a new system” containing all the decisions that complement the rest of the requirements. It is important to note that, as it itself proposes, with this methodology it is possible 1) to make use of catalogs of requirements and solutions and 2) to trace architecture and implementation decisions almost automatically.

Fig. 6. Safety-driven requirements elicitation methodology. Identification of safety requirements is performed in the early stages of requirement harvesting and is included in the final requirement specifications.

5 TRACEABILITY IN AN MDE FRAMEWORK

According to Gotel [22], Requirements Traceability refers to “the ability to describe and follow the life of a requirement, in both a forward and backward direction.” The purpose of forward traceability is to trace software artifacts that are created in the normal course of development. For its part, backward traceability seeks to link every software element to the artifacts involved in its creation. Software artifact traceability makes it possible to examine the system developed and its requirements in much greater detail, and also to determine any implications that a change may entail. These capabilities may be summarized as follows:

- Certify whether the different requirements have been considered and whether the resulting implementation is compatible with them.
- Verify whether the architectural model is compliant with the safety requirements.
- Ascertain the impact that modifying a requirement may produce.

In order to be able to save and later process the traceability links defined, we need to have a repository for links between the various software artifacts that are generated. The model-driven software development framework summarized in Fig. 4 defines a metamodel for traceability as described below.

5.1 A Metamodel for Traceability

The only way to integrate traceability in a MDE proposal is to define a metamodel with which to model the relevant relationships. The “Traceability Metamodel” shown in Fig. 7 was inspired by other works, such as [23]. The development framework described in the previous section includes a model of traceability between every two consecutive levels of abstraction, as shown in Fig. 4. For example, for the particular case of traceability between safety requirements and the V³CMM component model, the Link element of the metamodel is defined by:

- An Element Model (source), which references an element of the safety-based requirements metamodel.
- An Element Model (target), which references an element of the V³CMM.

Fig. 7. Traceability metamodel. Links connect one source with many targets. A CompositeLink serves to group any kind of AbstractLink.

Fig. 8 shows an example of elements traced between any two consecutive levels of abstraction (from requirements to design and from design to programming language elements, etc.). Note that it is possible to integrate the relationships between one and many target element models within a single Link. At the same time, an element may be the target of various links with different sources. The idea of CompositeLink is to be able to define different levels of granularity by grouping individual Links into other more complex ones in accordance with the overall purpose that they serve. The LinkType attribute makes it possible to categorize the existing relationships and distinguish the level of the process on which each Link is located.

Traceability models are created during the process of model transformation. The first traceability model (between safety requirements and V³CMM models) is created manually by the user. From the component model V³CMM to the Ada code that is generated, all links are created automatically as part of the source-to-target model transformation

Fig. 8. Examples of types of Links that can be created with the traceability metamodel defined.

process. In other words, at the same time as the instructions for creation of models closer to the implementation platform are executed, a record of the links between the elements involved is left in the traceability repository.

5.2 The Traceability Report

Once the traceability model has been generated, we can obtain a detailed report of the elements traced, using the *Safety-Requirements-Trace-Tool* (SRTT) that has been developed (see Fig. 4). This tool, which was developed using JET, generates the traceability report in HTML format, performing a M2T transformation on the existing traceability models. With this report, designers can examine the way

Trace Link		Source Link		Target Link		Composite?
Name	Description	Metamodel	Model Element	Metamodel	Model Element	
H13 - watchdog	This link refers to a Component, three ports and two interface					☒
H13 to Component	Add additional emergency stop mechanisms. Add external motion sensors (6.4 ANSI). Watchdog implementation.	safety	safety:hasHAZARD - origins => Error in the primary positioning system controller. Communications failure. Power failure - reactionMeasures => Emergency alarm. Power robot off. Emergency stop. - description => The primary positioning system does not stop - id => H13	v3cm	SimpleComponent - name => Watchdog	☒ Child of H13 - watchdog
H13 to Port	The Watchdog Unit Component will receive commands from the MMI through this port	safety	safety:hasHAZARD - origins => Error in the primary positioning system controller. Communications failure. Power failure - reactionMeasures => Emergency alarm. Power robot off. Emergency stop. - description => The primary positioning system does not stop - id => H13	v3cm	component:port - name => P_Wdog_Cmd	☒ Child of H13 - watchdog
H13 to Port	The Watchdog Unit Component will receive the signal for implementing the watchdog pattern.	safety	safety:hasHAZARD - origins => Error in the primary positioning system controller. Communications failure. Power failure - reactionMeasures => Emergency alarm. Power robot off. Emergency stop. - description => The primary positioning system does not stop - id => H13	v3cm	component:port - name => P_Wdog_Wdog	☒ Child of H13 - watchdog

Fig. 9. Traceability report generated by SRTT: from safety requirements to architectural design.

Trace Link		Source Link		Target Link		Composite?
Name	Description	Metamodel	Model Element	Metamodel	Model Element	
WatchdogComponent to UML	Transformation of the Watchdog Component					☒
Component to UML	Add a description here ...	v3cm	simpleComponent - name => Watchdog	uml	uml:Package - name => Watchdog_Component - qualifiedName => Watchdog_Component	Child of: WatchdogComponent to UML
				uml	uml:Class - name => Watchdog_Core - visibility => package - qualifiedName => Watchdog_Component::Watchdog_Core	Child of: WatchdogComponent to UML
				uml	uml:Class - name => Component_Facade_Watchdog - qualifiedName => Watchdog_Component::Component_Facade_Watchdog	Child of: WatchdogComponent to UML
Port to Class	Add a description here ...	v3cm	component:Port - name => P_Wdog_Wdog	uml	uml:Class - name => Port_Watchdog1 - qualifiedName => Watchdog_Component::Port_Watchdog1 - isAbstract => true	Child of: WatchdogComponent to UML

Fig. 10. Traceability report generated by SRTT: from safety requirements to architectural design.

in which a requirement has been taken into account in the automatic code generation process and receive information on the solution adopted, which in some cases will be a design pattern.

Fig. 9 a screen-shot of the use of the SRTT, where we can see the elements stored in "Traceability Model #1." A name and description is given for each link along with information on the source and target of the elements traced. The first row of the report shows a CompositeLink grouping all the information on the Links involved in the architecture solution for hazard H13. In this case, the Links are for the WatchdogUnit component and its ports and interfaces. Fig. 10 shows the traceability report for "Traceability Model #2," generated automatically as part of the transformation process from component model to object model. To simplify the presentation, only the matches between the WatchdogUnit component at the architecture level (see Fig. 2) and the elements generated following execution of the transformation to UML (see Fig. 3) are shown.

Listing 1 shows an extract from the ATL code used in that transformation. Note that there are three rules for translation of the elements: V³CMM Component, Port and Interface to UML Classes, and Interfaces. Each of these rules includes the corresponding instruction for automatic generation of the link, which is stored in "traceabilityModel2." The link corresponding to interfaces and ports is of the simple kind (Link) and stores the names of the linked elements. In the case of the component link, a CompositeLink is created as a composite element that includes the simple Links to the component, port, and interface that form part of the UML model generated in the course of the transformation.

```

module v3TOuml; — Module Template
create umlM:uml, traceabilityModel2:traz
from vcmM:vcm;

rule COMPONENT {
from f:vcm!SimpleComponentDefinition
to core:uml!Class(name<-f.componentName+'_Core'),
facade:uml!Class(name<-f.componentName+'_Facade'),
p:uml!Package(name<-f.componentName,
packagedElement<-f.port,
packagedElement<-thisModule.resolveTmp(f.port,lp)),
packagedElement<-Set{real,facade}),
scdTrace:traz!Link(name<-f.componentName,
target<-Set{core,facade,p}),
t:traz!CompositeLink(child<-scdTrace,
name<-f.componentName,
child<-f.ports->
collect(i|thisModule.PortTrace(i)))
}

rule PORT {
from f:vcm!Port
to p:uml!Class(name<-f.portName,
interfaceRealization<-f.providedInterface),
lp:traz!Link(name<-f.portName,
source<-f, target<-p)
}

lazy rule PortTrace {
from f:vcm!Port
to lp:traz!Link(name<-f.portName,
source<-f, target<-p)
}

rule INTERFACE {
from f:vcm!Interface
to i:uml!Interface(name<-f.interfaceName),
li:traz!Link(name<-f.interfaceName,
source<-f, target<-i)
}

```

Listing 1. ATL code excerpt of the V³CMM to UML M2M transformation, extended with traceability.

The last step in the process is to generate code from the preceding models using M2T transformations. Traceability from UML models to a programming language is currently restricted by a major technological limitation: compilers do not accept the format in which the models are stored (normally XMI files) as an input. This means extra work for traceability, since the target of a link cannot be a line of source code but must necessarily be an element of a model according to a predefined metamodel. To get round this problem and be able to use links to elements of the programming language (in this case Ada 2005), we decided to define a basic metamodel that would include the concepts: package, class, interface, method, and attribute. The models corresponding to this metamodel ("Traceability Model #3") are generated as part of the process of execution of the M2T transformations, which are extended with M2M transformations.

In short, the traceability report enables the user to tell what requirements have been considered, which architecture elements support each one, what object-oriented elements are associated, and finally what portions of the code have been implemented.

5.3 Benefits of Traceability

In this section, we introduce the main benefits accruing from the use of the traceability reports described above. These benefits are as follows:

- *Validation that all safety requirements are supported.*

This validation can be performed by means of the traceability report described above. The report shows that all the safety requirements considered are represented in one or other architectural element of the V³CMM model. If the aim is to check that a safety requirement has been considered in full, it is possible to show only those architectural elements that are relevant to that requirement. For example, one of the solutions chosen to deal with hazard H13 in the case study reported above was to "add additional emergency stop mechanisms." Through the traceability report, the analyst can check that the architecture solution is composed of the WatchdogUnit component and the interactions with the ControlUnit component.

- *Verification that the architectural model is compliant with the safety requirements.*

The traceability report offers analysts explicit matches between the safety requirements and each element of the architecture solution designed. To be able to perform this verification, analysts must have the patterns used to satisfy the safety requirements properly cataloged. These patterns are well-known solutions in the field of safety systems. To verify the requirements, it is necessary to have the traceability report and details of the patterns used in the transformation rules that have been defined, but verification of correctness is not immediate. In the example of hazard H13, to be sure that the component model deployed to deal with the hazard is the right one, availability must be correctly characterized in terms of reliable communications, the existence of degraded modes of operation, and the permanent availability of safe stops. These characteristics can be verified more readily using the trace information supplied by the SRTT.

- *Establishing the impact of changing a safety requirement.*

System users habitually suggest changes in the requirements during the course of the development process. The traceability report can help analysts to assess the impact of such changes before they are implemented. The first thing that the analyst needs to know is how many elements have to be modified at each level, bearing in mind that modifications are propagated downward to lower-level models. Then again, the analyst can equally verify the repercussions that changes in a requirement will have on other requirements that have been introduced. It is important in this respect to be aware of the compromises among the quality attributes defined in the system. The SRTT can help engineers to make these decisions, as it helps them to see more clearly the dependencies between the original requirements and the architecture decisions used to fulfil them. For example, incorporating the WatchdogUnit component to deal with hazard H13 may negatively affect performance, since this requirement basically depends on the existing communications load. This kind of decision-making can be enhanced by different proposals linking architecture solutions to quality attributes [24]. These proposals can be used to create preliminary definitions of the software architecture, analyze architecture decisions in the absence of code, and analyze various candidate architectures for the development of a system. The main purpose is to ascertain the consequences of architecture decisions in the light of the quality attributes. These attributes must be examined bearing in mind their interdependence with other attributes, for they are not independent but interact with one another by way of structural relationships imposed by the architecture.

6 RELATED WORK

The work presented here offers support for traceability of software artifacts linked to safety requirements. Obviously, the approach adopted in the case of teleoperated service robots can be extrapolated to other types of systems. Moreover, the utility of the proposed approach has been demonstrated by means of an illustrative example.

There are many initiatives and projects that integrate code design and generation tools in general. Specifically, real-time systems developers are progressively incorporating MDE into their development processes. Prominent among these are the SAE (*Society of Automotive Engineers*) *Architecture Analysis and Design Language* (AADL) standard [25], the AUTOSAR¹ open architecture, the European project OpenEmbeDD,² and Artisan.³

Then, there are a number of studies that have successfully tackled traceability within the MDE framework. In [19], an approach is presented for supporting automated generation of trace information, and also, a tool-supported technique is offered to facilitate trace acquisition by generating trace information automatically.

There are many approaches that face the problem of integrating safety in MDE. In [26], an approach based on

1. <http://www.autosar.org/>.

2. <http://www.openembedded.org/>.

3. <http://www.artisansoftwaretools.com/>.

the use of *System Modeling Language (SysML)* is proposed to consider safety aspects, identify risks, manage, and mitigate them. In order to evaluate a system, it must be broken down into its various concerns. By integrating these concerns into the model, it is possible to support other concerns (e.g., safety). This facilitates the implementation of traceability policies, and could be combined with a MDE approach. However, authors do not provide concrete tools for automatically applying transformations and requirements traceability.

Biffi et al. [27] describe a work-in-progress from a case study, where software quality can be measured and improved by using a MDE approach. The goal of this work is to provide decision makers in safety-critical domains with the most relevant information in a timely manner. No formalism is used in the requirement and traceability definition. The authors argue that using MDE benefits quality systems design; however, as in the case of [19], their proposal does not include a specific development framework, where automatic transformations can be achieved.

A related but different problem is to test whether the chosen solutions really meet the safety requirements. Most of the development tools used for embedded systems design provide some support for the analysis and formal verification of UML or SysML designs. To mention a few such tools, TOPCASED is a set of Eclipse-based open-source-model editors and transformation and formal verification tools that use SysML among other modeling languages. IBM Rhapsody and IBM Rational Tau tool suites include the "Rhapsody UML Verification Environment," which supports verification of safety and liveness requirements for embedded controllers. The Artisan design tools include extensive support for formal analysis of safety-critical systems. In this vein, it could be possible to interoperate with these tools and transform the V³CMM models into SysML models that could be analyzed by such tools.

Geensoft Reqtify offers a solution for managing requirements traceability and impact analysis across Hw/Sw projects life cycle. It is worth mentioning the interoperability of this tool with the most well-known modeling and software development environments. IBM Rational DOORS helps users capture, link, trace, analyze, and manage changes to the requirements of systems, and can be integrated with other IBM tools. Finally, Esterel SCADE is a well-known environment for the development of safety-critical embedded software in several domains (aerospace, defence, and rail transportation, etc.). This tool offers requirements management, model-based design, simulation, verification, certified code generation, and interoperability with other development tools and platforms.

Our work contributes the following features: 1) we provide a requirements metamodel that gives due consideration to the safety aspect in the development of teleoperated service robot applications, 2) we provide a component metamodel and a set of model transformations to (semi)automatically derive object-oriented implementations from component-based architectural descriptions, and 3) we provide a tool that generates a traceability report from the model transformations. These results offer developers an integrated environment, which is favorable for reuse of

the software artifacts generated in the safety systems construction process. Moreover, the inclusion of an environment for traceability of these artifacts enhances the quality of the systems that are built in that it facilitates both modification of these artifacts and analysis of the impact of the changes on the safety requirements.

7 CONCLUSIONS

The approach described in this paper focuses on tracing safety requirements to design, and design to implementation through different model transformations. We have shown the efficiency and value of combining a model-driven approach with traceability capabilities when designing safety-related systems. Following are some of the challenges that have to be faced when developing safety or mission-critical software [28]:

- Mastering complexity: software for safety systems may be complex in terms of architecture, components, and functionality.
- Avoiding multiple descriptions of software: the rewriting of software descriptions (requirements, software architecture, detailed design, and code) is both expensive and error-prone due to inconsistencies between them.
- Reducing lack of accuracy of specifications: the inherent ambiguity of textual-based languages is subject to interpretations.
- Avoiding manual coding: the major part of the software development life cycle is taken up by code testing.
- Finding errors as soon as possible: some errors are only encountered during software deployment and testing, where the cost is exponentially greater. Integration is a very late stage in the software process to truly exercise the software.
- Reducing the complexity of evolution: when a requirement has to be changed, all the artifacts have to be updated accordingly, and all verification tasks must be redone.
- Improving verification: verification is needed for safety systems, and consequently, has a considerable impact on the project timing and budget.

MDE is seen as a way to address the above challenges. The MDE tools provide the necessary rigor for efficient development of safety systems in several application domains. Formal models can be used to develop complex systems, potentially generating millions of lines of source code. Moreover, these models are at the core of all activities in the software development life cycle. Thus, verification and validation activities may be performed at the level of these models, that is in the early stages of the project. In short, MDE offers benefits for the development of safety systems. Our proposal helps developers to meet the above challenges by providing 1) a development framework that allows us to integrate safety requirements from the early stages of software development and 2) a tool that generates traceability reports. As proof of concept, we have used our approach in the development of the EFTCoR system. The

tool constitutes a valuable mechanism for managing changes in requirements.

The implementation of the EFTCoR system offered results which are sufficiently significant to highlight the benefit of implanting a tool for traceability management within a MDE framework. The system involved 60 safety-related requirements. Forty-five percent of these requirements did not affect the architectural design neither its possible evolution (such as, the addition of electrical/mechanical limiting elements, the specification of usage and signaling procedures, and the use of safety certified elements), while the remaining 50 percent did affect software architecture in one of the following ways: 1) about eight percent implied the addition of new sensors to the robot to monitor the system, 2) forty percent implied the addition (or extension) of some components, and finally, 3) about six percent implied the design and adoption of redundant nodes. In managing 2) and 3), we really got benefits of considering traceability, since they implied to precisely evaluate the impact of extending/modifying significant parts of the system architecture. In particular, type 3) involved the addition of components highly coupled with the already existing ones. It was very useful to quickly detect the impact of these requirements over the software architecture.

Our plans for further work include integrating additional design patterns into the development framework, and providing developers with a friendly framework that integrates all the involved tools in a single development system.

ACKNOWLEDGMENTS

This work has been partially supported by the Spanish CICYT Project EXPLORE (ref. TIN2009-08572).

REFERENCES

- [1] A. Iborra, D. Alonso, F. Ortiz, J. Franco, P. Sánchez, and B. Álvarez, "Design of Service Robots," *IEEE Robotics and Automation Magazine*, special issue on software eng. for robotics, vol. 16, no. 1, pp. 24–33, Mar. 2009.
- [2] C. Fernández, A. Iborra, B. Álvarez, J. Pastor, P. Sánchez, J. Fernández, and N. Ortega, "Ship Shape in Europe: Cooperative Robots in the Ship Repair Industry," *IEEE J. Robotics and Automation*, vol. 12, no. 3, pp. 65–77, Sept. 2005.
- [3] American National Standards Institute, *ANSI/RIA R15.06: American National Standard for Industrial Robots and Robot Systems Safety Requirements*, 1999.
- [4] B. Douglass, *Doing Hard Time: Developing Real Time Systems with UML, Objects, Frameworks and Patterns*. Addison Wesley Longman Publishing Co., Inc., 1999.
- [5] D. Alonso, P. Sánchez, F. Ortiz, J. Pastor, B. Álvarez, and A. Iborra, "Experiences Developing Safe and Fault Tolerant Tele Operated Service Robots. A Case Study in Shipyards," *Service Robot Applications*, pp. 159–182, InTech Education and Publishing, 2008.
- [6] B. Selic, "The Pragmatics of Model Driven Development," *IEEE Trans. Software Eng.*, vol. 20, no. 5, pp. 19–25, Sept./Oct. 2003.
- [7] T. Mens and P. van Gorp, "A Taxonomy of Model Transformation," *Electronic Notes in Theoretical Computer Science*, vol. 152, pp. 125–142, 2006.
- [8] D. Alonso, "Software Development for Service Robots: A Model Oriented and Component Based Approach," PhD dissertation, Information Technologies and Comm. Dept., Technical Univ. of Cartagena, Apr. 2008.
- [9] IEEE Computer Society, *IEEE 610.12 1990 Standard Glossary of Software Engineering Terminology*, 1990.
- [10] H. Bruyninckx, "Robotics Software: The Future Should be Open," *IEEE Robotics and Automation Magazine*, vol. 15, no. 1, pp. 9–11, Mar. 2008.
- [11] A. Clifton and I. Ericson, *Hazard Analysis Techniques for System Safety*. Wiley, 2005.
- [12] European Committee for Electrotechnical Standardization, EN 61508: Functional Safety of Electrical/Electronic/Programmable Electronic Safety Related Systems 2003.
- [13] D. Schmidt, M. Stal, H. Rohnert, and F. Buschmann, *Pattern Oriented Software Architecture: Patterns for Concurrent and Networked Objects*, vol. 2. Wiley, 2000.
- [14] B. Douglass, *Real Time Design Patterns: Robust Scalable Architecture for Real Time Systems*. Addison Wesley Professional, 2002.
- [15] C. Szyperski, *Component Software: Beyond Object Oriented Programming*, second ed. Addison Wesley Longman Publishing Co., Inc., 2002.
- [16] D. Steinberg, F. Budinsky, M. Paternostro, and E. Merks, *EMF: Eclipse Modeling Framework*, second ed. Addison Wesley Professional, 2008.
- [17] F. Jouault, F. Allilaire, J. Bézuvin, and I. Kurtev, "ATL: A Model Transformation Tool," *Science of Computer Programming*, vol. 72, nos. 1/2, pp. 31–39, 2008.
- [18] J. Nicolás, J. Lasheras, A. Toval, F. Ortiz, and B. Álvarez, "An Integrated Domain Analysis Approach for Teleoperated Systems," *Requirements Eng. J.*, vol. 14, no. 1, pp. 27–46, Feb. 2009.
- [19] A. Egyed and P. Grünbacher, "Automating Requirements Traceability: Beyond the Record & Replay Paradigm," *Proc. 17th IEEE Int'l Conf. Automated Software Eng.*, pp. 163–171, 2002.
- [20] V. Vicente Chicote, B. Moros, and A. Toval, "REMM Studio: An Integrated Model Driven Environment for Requirements Specification, Validation and Formatting," *J. Object Technology*, vol. 6, no. 9, pp. 437–454, Oct. 2007.
- [21] A. Toval, B. Moros, J. Nicolás, and J. Lasheras, "Eight Key Issues for an Effective Reuse Based Requirements Process," *Int'l J. Computer Systems Science and Eng.*, vol. 23, no. 6, pp. 1–20, Nov. 2008.
- [22] O. Gotel, "Contribution Structures for Requirements Traceability," PhD dissertation, Dept. of Computing, Imperial College, 1995.
- [23] D. Kolovos, R. Paige, and F. Polack, "On Demand Merging of Traceability Links with Models," *Proc. Second European Conf. Model Driven Architecture (EC MDA) Workshop Traceability*, July 2006.
- [24] R. Kazman, L. Bass, and M. Klein, "The Essential Components of Software Architecture Design and Analysis," *J. Systems and Software*, vol. 79, no. 8, pp. 1207–1216, Aug. 2006.
- [25] P. Feiler, D. Gluch, and J. Hudak, "The Architecture Analysis & Design Language (AADL): an Introduction," Technical Report CMU/SEI 2006 TN 011, Software Eng. Inst. (SEI), Carnegie Mellon Univ., <http://www.sei.cmu.edu/pub/documents/06-reports/pdf/06tn011.pdf>, 2006.
- [26] M. Hause and F. Thom, "An Integrated Safety Strategy to Model Driven Development with Sysml," *Proc. Second Institution of Eng. and Technology Int'l Conf. System Safety*, pp. 124–129, 2007.
- [27] S. Biffi, R. Mordinyi, and A. Schatten, "A Model Driven Architecture Approach Using Explicit Stakeholder Quality Requirement Models for Building Dependable Information Systems," *Proc. Fifth Int'l Workshop Software Quality (WoSQ '07): Int'l Conf. Software Eng. (CSE) Workshops 2007*, p. 6, 2007.
- [28] J. Krasner, *Model Driven Development of Certifiable Software: A Best Practice for Safety Critical Applications*. Am. Technology Int'l, Inc., Dec. 2008.

Pedro Sánchez received the PhD degree in computer science from the Technical University of Valencia, Spain, in 2000. Since 1996, he has participated in various projects dealing with software engineering and conceptual modeling. He joined the Division of Systems and Electronic Engineering (DSIE) in 2000. He is currently an associate professor at the Technical University of Cartagena in the field of computer science. His current research interests include model-driven engineering for real-time systems.

Diego Alonso received the BSc and MSc degrees in industrial engineering from the Technical University of Valencia in 2001 and the PhD degree from the Technical University of Cartagena in 2008. Currently, he is a lecturer in the Information Technologies and Communications Department, Technical University of Cartagena. He has published more than 20 refereed journal and conference papers. His research interests focus on the application of the model-driven engineering approach to the development of component-based reactive systems with real-time constraints. He is a member of the International Federation of Automatic Control (IFAC).

Francisca Rosique received the BSc and MSc degrees in telecommunication engineering from the Technical University of Cartagena in 2007. Currently, she is a lecturer in the Information Technologies and Communications Department, Technical University of Cartagena, where she is working toward the PhD degree. Her research interests focus on the application of the model-driven engineering approach to the development of reactive systems.

Bárbara Álvarez received the MSc degree in telecommunication engineering from the Technical University of Madrid, Spain, in 1993, and the PhD degree in telecommunication engineering from the Technical University of Madrid in 1997. In 1998, she was a faculty member at the University of Murcia Spain, and in 1999, at the Technical University of Cartagena, Spain. Currently, she is an associate professor in the Department of Information and Communication Technologies, Technical University of Cartagena. She has published more than 50 refereed journal and conference papers. Her research interests cover robotics and software engineering. She is a member of the Ada-Europe, the International Federation of Automatic Control (IFAC), and the IEEE.

Juan A. Pastor received the MSc degree in telecommunication engineering from the Technical University of Madrid, Spain, in 1997, and the PhD degree in telecommunication engineering from the Technical University of Cartagena, Spain, in 2002. Currently, he is an associate professor in the Department of Information and Communication Technologies, Technical University of Cartagena. He has worked as a research engineer in robotics for nuclear power plants from 1995 to 2000. In 2000, he joined the Division of Systems and Electronic Engineering (DSIE). His research interests include mechatronic systems design and analysis, robotics, design patterns, and model-driven development.

► For more information on this or any other computing topic, please visit our Digital Library at www.computer.org/publications/dlib.

Impact Factor Trend Graph: IEEE TRANSACTIONS ON COMPUTERS

Click on the "Return to Journal" button to view the full journal information.

***Impact Factor -- see below for calculations**

The journal impact factor is a measure of the frequency with which the "average article" in a journal has been cited in a particular year. The impact factor will help you evaluate a journal's relative importance, especially when you compare it to others in the same field. For more bibliometric data and information on this and other journal titles click on the "Return to Journal" button.

NOTE: Title changes and coverage changes may result in no impact factor for one or more years in the above graph.

2010 Impact Factor

Cites in 2010 to articles published in: 2009 = 134	Number of articles published in: 2009 = 133
2008 = 292	2008 = 132
Sum: 426	Sum: 265

Calculation: $\frac{\text{Cites to recent articles}}{\text{Number of recent articles}} = \frac{426}{265} = \mathbf{1.608}$

2009 Impact Factor

Cites in 2009 to articles published in: 2008 = 191	Number of articles published in: 2008 = 132
2007 = 310	2007 = 143
Sum: 501	Sum: 275

Calculation: $\frac{\text{Cites to recent articles}}{\text{Number of recent articles}} = \frac{501}{275} = \mathbf{1.822}$

2008 Impact Factor

Cites in 2008 to articles published in: 2007 = 265	Number of articles published in: 2007 = 143
2006 = 474	2006 = 140

Sum: 739
Calculation: Cites to recent articles 739 = **2.611**
Number of recent articles 283

Sum: 283

2007 Impact Factor

Cites in 2007 to articles published in: 2006 =194 Number of articles published in: 2006 =140
2005 =273 2005 =138
Sum: 467 Sum: 278

Calculation: Cites to recent articles 467 = **1.680**
Number of recent articles 278

2006 Impact Factor

Cites in 2006 to articles published in: 2005 =121 Number of articles published in: 2005 =138
2004 =267 2004 =134
Sum: 388 Sum: 272

Calculation: Cites to recent articles 388 = **1.426**
Number of recent articles 272

[Acceptable Use Policy](#)
Copyright © 2012 [Thomson Reuters](#).

THOMSON REUTERS

Published by Thomson Reuters

Rank in Category: IEEE TRANSACTIONS ON COMPUTERS**Journal Ranking**

For **2010**, the journal **IEEE TRANSACTIONS ON COMPUTERS** has an Impact Factor of **1.608**.

This table shows the ranking of this journal in its subject categories based on Impact Factor.

Category Name	Total Journals in Category	Journal Rank in Category	Quartile in Category
COMPUTER SCIENCE, HARDWARE & ARCHITECTURE	48	12	Q2
ENGINEERING, ELECTRICAL & ELECTRONIC	247	63	Q2

Category Box Plot

For **2010**, the journal **IEEE TRANSACTIONS ON COMPUTERS** has an Impact Factor of **1.608**.

This is a box plot of the subject category or categories to which the journal has been assigned. It provides information about the distribution of journals based on Impact Factor values. It shows median, 25th and 75th percentiles, and the extreme values of the distribution.

A framework for developing home automation systems: From requirements to code

Pedro Sánchez*, Manuel Jiménez, Francisca Rosique, Bárbara Álvarez, Andrés Iborra

Systems and Electronic Engineering Division (DSIE), Technical University of Cartagena, Campus Muralla del Mar s/n, 30202, Cartagena, Spain

ARTICLE INFO

Article history:

Received 28 June 2010

Received in revised form

14 December 2010

Accepted 19 January 2011

Available online 1 February 2011

Keywords:

Home automation

Model driven

Code generation

ABSTRACT

This article presents an integrated framework for the development of home automation systems following the model-driven approach. By executing model transformations the environment allows developers to generate executable code for specific platforms. The tools presented in this work help developers to model home automation systems by means of a domain specific language which is later transformed into code for home automation specific platforms. These transformations have been defined by means of graph grammars and template engines extended with traceability capabilities. Our framework also allows the models to be reused for different applications since a catalogue of requirements is provided. This framework enables the development of home automation applications with techniques for improving the quality of both the process and the models obtained. In order to evaluate the benefits of the approach, we conducted a survey among developers that used the framework. The analysis of the outcome of this survey shows which conditions should be fulfilled in order to increase reusability.

© 2011 Elsevier Inc. All rights reserved.

1. Introduction

Rapid advances in electronics, information and communications technology (leading to miniaturization and improvement of performance of computers, sensors and networking) have given rise to the development of several home automation (HA) technologies (Chana et al., 2009). HA applications integrate comfort, energy saving, security and communications functions. The aim of an HA system is to provide homes with a certain degree of 'intelligence' and to improve the quality of life of its inhabitants. Tasks like automatically switching lights and heating, cutting off the supply when gas or water leaks are detected or controlling the home devices remotely from a mobile or a computer through an Internet connection are typical applications of HA domain.

There are several HA standards and protocols adopted by the leading companies in the market. Some notable examples are KNX (ISO/IEC14543-3-X and EN50090 standards), Lonworks (ISO/IEC 14908, EN14908 and EIA-709-1 standards) and X10 (a well known international and open industry standard for communication among electronic devices). However, one of the main problems of HA development resides in the fact that there is no consensus in the standard to implement these applications. As stated in Miori et al. (2006), it is improbable that there will be a single dominant technology for HA in short term. Furthermore, each of such stan-

dards provides its own software suite to create HA applications and program the devices in question. Hence the particular technology (specific platform) must be selected at the initial design stage, as much as the tools and devices to be used depend on this choice. These facts make the development of HA applications strongly platform dependent, making it very difficult to raise the abstraction level and work with HA domain concepts rather than technology elements.

This drawback can be avoided by adopting the well known Model-Driven Development technique (MDD) (Selic, 2003). In this approach, application code can be automatically generated from platform-independent models. Although MDD techniques have been developed some years ago, there are no well known integrated frameworks for developing HA systems. However, there is currently a need for the creation of tools to develop these systems. These tools should allow the generation of code for several platforms. In this work, we present an integrated framework that allows the definition of HA systems at different levels of abstraction, from requirements to code. Taking advantage of using a domain specific language (DSL) (Mernik et al., 2005) the developer can work with graphical elements and concepts of the HA domain.

DSLs provide easy, intuitive domain-specific descriptions of systems using graphical or textual models. A DSL includes the tooling infrastructure for creating and transforming models into executable instances of the language (Kelly and Tolvanen, 2008). In this context, the appearance of the MDD approach has increased the research on these languages as well as new automatic code generation techniques. Nevertheless, the development of DSLs is very time

* Corresponding author. Tel.: +34 968326460; fax: +34 968325973.

E-mail address: pedro.sanchez@upct.es (P. Sánchez).

consuming: ideally, models made using a particular DSL should be able to be reused across several implementations to amortize this effort. The reuse of DSL models across different development projects can help reduce the cost of these projects.

Several works discuss advantages and drawbacks of using DSLs. Maintenance, flexibility, productivity, reliability and reusability are attributes commonly found in these types of languages (Hermans et al., 2009). With DSLs, reuse is feasible at the model level, making it possible to reuse partial or entire models, rather than pieces of platform-dependent code. Thus, the beginning of a new software development project can be done from existing reusable assets. A surprising fact is that reuse hardly plays a significant role in current DSLs as demonstrated in Hermans et al. (2009) for the study of a particular DSL.

We identify two key aspects that determine the feasibility of reuse in the context of DSLs: (1) to select a model or a model fragment for reuse you must know what is does; and (2) to achieve effective reuse, you must be able to discover the model fragment faster than you could build it. Besides, the use of best practices for DSL definition and implementation determines the success in model reuse. For instance, language creators usually try to avoid modeling errors by imposing dozens of strongly enforced integrity rules that prevent modelers from temporarily breaking the rules while they are trying to reuse their models. Moreover, interconnected models should have minimal coupling to improve modularization and avoid data duplication which lead to maintenance and reuse problems.

In short, this article contributes to the state of the art with the following features:

- A framework that integrates a set of tools for defining HA applications at different levels of abstractions.
- A set of model transformations (Mens and van Gorp, 2006) that enables developers to get full executable code.
- Traceability capabilities (Ramesh and Jarke, 2001) to improve quality both of the process and of the models obtained.
- A survey that demonstrates the success of reusing models in MDD by means of generic requirements. We investigate factors that contribute to this success.

The article is structured as follows: Section 2 deals with introducing the basis of the proposal and the related works. Section 3 presents the proposed framework and also offers a general overview of the implementation using Eclipse. Section 4 explains the developed tool for managing traceability. Section 5 gives a cost model of the approach. Section 6 details an evaluation of the approach based on a survey and a comparison of the developed tool with two HA commercial tools. Finally, Section 7 is dedicated to conclusions and future work.

2. Foundations and related work

2.1. Home automation systems development

At the present time, developers of HA applications mainly use software tools provided either by the device manufacturer, in the case of proprietary system, or by the associations responsible for providing support for the technology in the case of the standard systems. These tools are usually platform-dependent, code generation-oriented integrated environments which do little to raise the level of abstraction. Moreover, the concrete syntax that they use is not usually very intuitive, so that the user requires very specialized training and can only work in the immediate context of the solution.

The whole process of development of HA applications is carried out by an expert in the domain who collates the customer's requirements for an installation (elements to be integrated, services required, selection of a concrete technology, etc.) based on his own experience. This expert carries out the selection and deployment of the devices and afterwards programs them (using a platform-dependent development infrastructure) so as to achieve the desired functionality. Working in this manner it is rather difficult to achieve some of the desired attributes of software systems such as interoperability, flexibility, re-use and productivity. Besides, tools to develop projects are completely different in each platform, so learning a new technology implies new training. Thus, developers usually focus on a particular technology, leaving aside other platforms. This is due to the long training time and high specialization required (a good developer would need to have undertaken around 100 h of training and have months of practice).

2.2. Related work

2.2.1. MDD for HA development

The literature offers a few examples of works which try to reach in an integrated way the development of HA systems using an MDD approach. Among these it is important to highlight the works of Muñoz et al. (2006), Voelter and Groher (2007) and Nain et al. (2008) that outline the necessity of using a model driven approach in HA systems development. The aim is to increase the level of abstraction, the productivity and the quality of the software, besides maintaining the independence of the implementation platform. These proposals represent a good example of the advantages that the use of MDD offers in the development of HA systems, but they also present some drawbacks. In the first place, Muñoz uses the UML notation for requirement elicitation which is not very intuitive for experts in the field of HA. In the work of Voelter, a set of HA devices is defined in the meta-model. Applications are created using the tool named Tree Editor provided by the plug-in EMF for Eclipse. Hence to use an HA device not included in the meta-model, it is necessary to build a new meta-model or extend the existing one. Nain presents EnTiMid as a middleware composed of several layers. A driver layer is in charge of the connection between the devices and the Unified Service layer. A bridge layer links the Unified Service instances to diverse service technologies such as Universal Plug And Play (UPnP) and Devices Profile for Web Services (DPWS) (Jammes et al., 2005). The work defines EnTiMid as a middleware implementation that supports various services access models and also describes how these artifacts are generated using MDD.

In these proposals the code generation is oriented to obtain OSGi (Open Service Gateway Initiative) drivers for a server or middleware platform, and not to the programming of the HA devices. Therefore, it will always be necessary an expert of the specific platform to program these devices.

Contrary to the previous examples, in our framework the level of abstraction and usability of requirements modeling rises with the use of a DSL that uses specific concepts of the domain. In addition, our proposal guides the code generation to the automatic programming of the devices of the selected HA technology. In this way the need for specific knowledge of each platform is avoided, as well as the intervention of an expert in the technology.

2.2.2. Reuse of DSL models

The literature distinguishes between two general types of reuse approaches (von Kethen et al., 2002): composition and generation-based approaches. Composition-based approaches are based on composing reusable assets. This type of approach is typically applied for design or code reuse. Generation-based approaches focus on instantiating reusable abstractions. Popular

examples are product-line approaches (Clements and Northrop, 2002) or patterns (Lam, 1998). For requirements reuse, generation-based approaches are more popular. A detailed survey about product line based requirements reuse can be found in Bühne et al. (2006). In Jacob and Reed (2000) a summary of general approaches for requirements reusing is given. Others have analyzed the reuse of requirements in domains where a set of requirements are frequently used (Lam et al., 1997). All these works have in common the interest in reuse as a key to improving software development productivity and quality.

Nowadays there is also a considerable interest in evaluating the use of DSLs in practice. In DSM (2010), there are some examples and case studies of how the use of DSLs has been applied in many industrial applications (mobile applications, business process, embedded systems, etc.), consistently improving productivity by 10 times. Several authors demonstrate that the use of DSLs raises the level of abstraction beyond coding and consequently make development faster and easier. For example, in Kärnä et al. (2010) an approach applied to sports instruments and heart rate monitoring devices is described. The evaluation of this approach provides an increase of at least 750% in developer productivity, and what is more, a great improvement in the quality of the code and development process. Since DSLs are based on the concepts already known and used in the domain, these constructs are easier to understand and remember for all developers, promoting the reuse of validated solutions (Kelly and Tolvanen, 2008).

All the previous works concentrate either on requirements reuse or in the use of DSLs but none of them analyzes the benefits of combining both approaches. Furthermore, there is little evidence in the literature to suggest that requirements reuse is widely practiced in DSL-based proposals. At the same time, the reuse of requirements and models in the context of MDD applied to DSLs is also in an initial stage.

3. A framework for HA systems development

Fig. 1 shows the main layers and elements involved in the use of the proposed framework for the development of HA systems following the MDD approach. As can be seen, the different MDD levels correspond to: HA requirements, domain specific descriptions, UML-based component level, and executable code for specific platforms. The intermediate UML-based component level allows developers the integration of other types of applications such as robotics and computer vision. Each model is built or obtained according to its meta-model by following the MDD paradigm.

Fig. 2 illustrates the steps involved in the use of the framework. The different abstraction levels correspond to requirements, DSL (using a predefined catalogue of functional units as detailed

Fig. 1. Overview of the framework.

at (Jimenez, 2009)), and executable code for specific platforms (in our case, KNX/EIB for deploying HA systems). The correspondences between the requirements and the DSL level are established manually. In this way, the partial solutions for each HA requirement are catalogued. When building a new application, the user may inspect this catalogue identifying the requirements that are going to be used. By doing this it is possible to reuse the solutions adopted with the DSL in the development of previous applications. From the DSL level towards code generation, the transformations are carried out automatically. Each model-to-model transformation is registered in the corresponding traceability model that is later processed by a tool that provides the user with a set of reports (as will be described later in Section 4).

As demonstrated in Section 6, the existence of a 'generic' requirements catalogue for subsequent system instantiation can significantly contribute to model reuse. Modelers select a subset of these generic requirements when developing a new system. For each generic requirement a model fragment is given using the DSL. A model fragment is part of a complete model in the sense that accomplishes part of the desired functionality. Then, reuse (see Fig. 3) may be fulfilled by the integration of all the model fragments into the system model to be developed. It is possible that a model fragment would be syntactically or semantically incomplete. Thus, integrity rules should be disabled temporarily in order to facilitate the integration of these model fragments.

The following aspects are required in order to achieve a framework with such functionality: (1) to develop graphical user

Fig. 2. A scheme of the methodology.

Fig. 3. Overview of the approach: from generic requirements to integrated models.

interface editors, and (2) to define and implement transformations between the defined layers from DSL models to code. All these tools have been implemented by using the Eclipse development framework. The following subsections give the details of the solutions adopted in each layer.

3.1. Requirements management

In order to integrate requirements management in the MDD process it is necessary to adopt a meta-model that allows developers to catalogue them. At the present time it is possible to find different requirements meta-models in the literature (Vicente-Chicote et al., 2007; Molina and Toval, 2009) with different assumptions and relationships. Such meta-models improve requirements reuse and serve as structured requirements reference models. In this work, we have decided to use a basic meta-model that has enough expressiveness to represent requirements. Fig. 4 shows the class diagram corresponding to the meta-model proposed.

This meta-model is integrated in the development environment and it has the level of complexity and sufficient versatility to be able to adopt an approach of requirements reuse. This meta-model is valid for other domains and, at the same time, simple in its structure. Thus, we decided to implement an Eclipse-compliant requirements meta-model editor from scratch in order to facilitate the integration with the rest of software modules. The root element is called *Catalogue* where a group of requirements (*Require-*

ment) is integrated. Each one of these requirements includes a name, a description and the validation procedure that the user should adopt to check that the deployed system has the desired capability. For example, given the requirement “there should be smoke detection”, the validation procedure would establish “provoke a simulated situation with smoke and verify that alarms go on”. The requirements are not isolated, instead they are related to the whole. To this end the element *Relationship* allows the connection of a requirement (*source* relationship) with one or more destination requirements (*target* relationship). The type corresponding to the relationship among requirements is identified with the element *TypeOfRelationship*. Table 1 shows an excerpt of a catalogue of requirements for HA systems keeping in mind the structure of this meta-model.

As it can be seen relationships have been considered in the cases of requirements 3, 5, 8 and 9. In some cases the presence of a specific requirement is required (for example, requirement number 5 requires number 4), at other times at least one of those indexed is required (as in the case of requirement number 3 with regard to numbers 1 and 2). The relationship can also be of inclusion (see requirement number 9) where in this case the requirement is indicated as being of a greater conceptual level. The relationship type among requirements is not closed and depends on the user. Nevertheless, this factor should be kept in mind because of the implications when deriving the specified models with the DSL. The matches between each requirement and fragments of the DSL model are traced. By proceeding in this manner, the specification at the DSL level of a new application starts by reusing model fragments. Finally, all the model fragments (reused and new ones) are integrated into a single model to undergo the successive transformations to code.

3.2. Domain specific descriptions

The framework includes a DSL (see (Jimenez, 2009) for a detailed description) for modeling HA systems in a platform-independent way. Such a DSL provides easy and intuitive descriptions of HA systems using graphical models. Any HA system incorporates a number of elements (called *Functional Units*), which are present in all the technologies and standards relevant to the domain. These differ in their architecture and the protocols they use, but they are identical when it comes to capability. In order to encourage the re-use of these functional units and to avoid having to repeatedly define the same unit for each application, it was decided to use

Fig. 4. Basic requirements meta-model.

Table 1
Example of some of the catalogued requirements.

Id	Name	Description	Validation procedure	Targets	Type
1	Light power on/power off	Control of power on/power off of a light (from any HA device)	Activate the switch and check that it turns on/turns off the light controlled by this switch	None	None
2	Lighting level regulation	Regulation of the intensity of a light	Check that when the lighting level regulator is activated the light increases/decreases correctly	None	None
3	Light automatic power on/power off	Power on/power off of a light automatically when presence is detected	Enter the room and check the lights switch on automatically	1, 2	Includes one or more
4	Blind raising/lowering	Control of the raising/lowering of a blind from any device	Activate the corresponding switch and check that the blind rises/lowers	None	None
5	Blind automatic raising/lowering	The raising/lowering of a blind automatically according to external light intensity	Check that in the evening the blinds lower automatically	4	Includes
6	Burglar alarm	Detection of intruders through the use of presence detectors	Activate the alarm and check that this alarm goes off when entering in the building	None	None
7	Climate control	Control of temperature increase/decrease	Indicate a temperature and check that this temperature is reached	None	None
8	Automatic climate control	Automatic temperature increase/decrease according to timetables and presence	Check that in the absence of people in the building the temperature reaches $\pm 4^{\circ}\text{C}$ of the instructed temperature	7	Includes
9	Energy saving	Energy consumption saving	Check energy consumption	3, 5, 8	Includes all of them
10	Smoke alarm	Detection of smoke	Generate smoke and check that the alarm is triggered	None	None
11	Flood alarm	Detection of water leaks	Wet the detector and check that the alarm is triggered	None	None

a catalogue of reusable functional units. For their part, functional units have some *Services* through which they are able to interact with other units. Many of these services are repeated among the functional units, so that a catalogue of services is available with *definitions of services* that can be reused in any functional unit. Basically, any HA application is described with the DSL by means of the instantiation of functional units that are predefined in a catalogue. These instances can be configured by adding the necessary values to their parameters. The links between functional units indicates, through services, the way in which these functional units will interact with the rest of the system. All the functional units have an associated location in a Building Structure View. This view has been taken into account in the DSL meta-model definition. Since the graphical editor is currently under development, this view is created using a hierarchical tree-based editor (Eclipse Tree Editor).

For implementing the DSL tool we have used the Eclipse Graphical Modeling Framework (GMF), which helps to automatically generate graphic editors as Eclipse plug-ins from models. The tool provides an integrated environment to design HA applications with the DSL. Fig. 5 shows a screenshot of some of the graphical elements that can be dragged to the drawing area from the palette of the tool.

Fig. 6 illustrates a snapshot of the developed tool with the part of the DSL model corresponding to an application of lighting management with comfort and energy saving functions in a meeting room. To achieve energy saving and comfort, the system activates a power-off function when it detects no presence in the room after 5 min. The functional unit icons suggest their meanings. The complete specification is given by the model's graphical view plus the parameterization of the corresponding functional units properties. In this example, one push button controls three lighting points in the room. Push button (PB-1) switches and dims the lights LDM-1 to LDM-4. To model this behaviour, the push button has been connected to a dimming input controller (DMI-1) using a channel link (red line) that binds required (PBactivated) and provided

(DMIactivated) services (both services must be instances of the same service definition). At the same time, lights are linked to their controllers (DMO-1 to DMO-4), which switch and dim the lights. Finally, controller services are associated with logical links (dashed green lines). Presence detector PIR3-1 is interconnected through a channel to the functional unit acting as controller (SWI-1). This controller calls the TMtempIn service from the timer (TM-1) every time the system detects a presence. So, the timer switches the lights on when the system detects a presence and switches them off after 5 min of detecting no presence. (For interpretation of the references to color in this text, the reader is referred to the web version of this article.)

As Fig. 6 shows, the tool also includes an area where the available properties (attributes, parameters, etc.) are displayed and can be modified for the selected element (functional unit or link). Properties are shown using editable text fields or drop-down lists to provide an intuitive and easy access to them.

A second case study example can be seen in Fig. 7. It shows the floor plan and the home automation devices needed to meet the following requirements given by the client: *"I want to have lights that I can regulate in my living room and dining room. In the patio and the balcony I'm concerned about security as it is easy to break in. During the night we enjoy staying outside so if lights switched on automatically that would be neat. My wife often forgets to turn off the stove in the kitchen and it gets filled up with smoke, if you could do anything about it that would be great. I want the shutters of the bedrooms to go up automatically in the morning and down at night. Last year we had a broken pipe at the bathroom and we got both levels of the house flooded, I want to have something that tells me if there is a flood but just in the two bathrooms from the upper floor. Finally I would only accept the installations if you could do it without reinstalling cable in the whole house, perhaps with some wireless gadget. Do you want to join us on Friday? Every Friday at 9:00 pm we play poker at the living room"*.

Instance of functional unit: includes a representative icon, its name, and parameters and the actual value for these parameters.

Link between functional units. A link is displayed as a red line when it is a physical channel. Otherwise the link is displayed as a discontinuous green line with end points. In these links, three labels are shown, one for each service that participates in the link and a third that shows the definition of the service (centre label).

Scene: contains the steps that will be performed when this starts. A scene step shows the service that will be performed and the icon of the functional unit that it belongs to.

Fig. 5. Graphical representation and description of some of the elements used in the graphical language.

Fig. 8 shows part of the DSL model that resulted from integrating DSL fragments corresponding to some requirements. As DSL models for each requirement are created it can be noticed that the same device can be used for several requirements. So an overlapping effect may occur. For example lights that turn On/Off by a push-button, can also be turned On/Off by a presence sensor or by a weekly timer. The same presence sensor that turns the lights On/Off also switches an alarm and sends an SMS to a mobile when it is activated. It turns out to be the same alarm and mobile that is used by the smoke detection and flood detection (as shown in Fig. 8). There can be even more devices overlapping as the installation increases, making it even harder to keep a trace of the correspondences between devices and requirements. Therefore, classifying which devices belong to which requirements becomes a complex and important task.

Both the applications and the catalogue of functional units and services may be modified and extended without modifying the underlying DSL or tools (editor, meta-models and transformations).

For example, to add a home entertainment system it should be added to the list of functional units with the required/provided services. Those services that were not already defined would be added to the service catalogue. Once the catalogue has been expanded, this device could be used for new applications or extensions of the existing ones.

Other proposals, such as (Muñoz et al., 2006; Voelter and Groher, 2007; Nain et al., 2008), define devices within the meta-model itself, so that the addition of new devices involves modifying the meta-model and thus the transformations rules and other artifacts involved in the code generation process.

3.3. Graph grammar transformations and code generation

Transformations between the DSL and the component-based lower layer are completely defined using a graph grammar-based approach (Rozenberg, 1997) (in particular the EMT plug-in (Biermann et al., 2006) for the Eclipse environment). The fact that

Fig. 6. A screenshot of the tool including the DSL model for a lighting application with comfort and energy saving functions.

Fig. 7. House plan with devices included in it and a descriptive table of the considered devices.

models are usually represented by graphs makes a graph grammar more attractive than other approaches; for instance, transformation rules expressed by means of graphs are easier to understand and trace. The overall conversion from the DSL to the component model uses several transformations. Each transformation is expressed with rules with a left hand side (LHS), representing

the pre-conditions for the transformation, and a right hand side (RHS), showing the result of the transformation. Both the LHS and RHS are graphs, both of which usually contain elements from both the input and output models. The inclusion of elements from the output model in the LHS ensures that the transformations are executed in the correct sequence. A rule may also have a Boolean

Fig. 8. DSL fragments for various requirements needed within the previous application.

Fig. 9. Example of graph transformation rule: from DSL to component model level. Black: catalogue view (DSL) instance; red: target (component model) instance; blue: application view (DSL) instance; orange: transformation instance. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of the article.)

negative application condition (NAC), which must not be satisfied to apply it. To apply a rule to a source graph (the graph to be transformed), there must exist a sub-graph isomorphism from the LHS to the source graph. After the application, there must be a sub-graph isomorphism from the RHS to the result graph.

Fig. 9 shows an example of a graph transformation rule. The rule states that when a service (LHS) is found it must be transformed into ports, interfaces and services of the target component model (RHS). However, this rule is not applied if the transformation has already been performed (NAC). The result of applying all the rules to the DSL fragment of Fig. 6 is the UML-based component model shown in Fig. 10. For example, *StdFUIInstances* of Fig. 6 are transformed into the *SimpleComps* shown in Fig. 10. For a complete description of the DSL to component model graph transformation rules and the considered meta-models see (Jimenez, 2009).

By means of additional graph grammar transformations, the components are transformed into executable models for the selected platform. In this regard, a target platform must first be selected. Two key points have been identified for selection of this platform: (1) the technology must be supported by international

standards and (2) the tools for programming the devices must be available and able to be interfaced externally. The two leading HA technologies mentioned above (KNX and Lonworks) fulfill these requirements.

As our research group has a wide experience in KNX, this has been selected as the first platform-specific infrastructure. For this last step, a meta-model has been defined for the KNX/EIB technology which considers the Domain Object Model (DOM) used by the commercial ETS tool (Engineering Tool Software). Platform specific models are independent of specific commercial tools and serve as the source for model-to-text transformations. JET (EC, 2010) and the macro editor called ITTools (IT, 2010) have been chosen for implementing these transformations. The ITTools plug-in allows us to interface to the manufacturer's environment using the VBScript programming language. In other words, it is a KNX/EIB specific tool which allows the creation of programming code for this platform by means of a scripting language, so the reuse of platform-specific tools is promoted.

Below an excerpt of a JET template is shown. It is used to generate the ITTools macro needed to insert new devices in the HA project:

Fig. 10. UML-based component model obtained after applying the graph transformation rules to the DSL example of Fig. 6.

Fig. 11. ETS Project obtained after execution of the ITTools macros for the example of Fig. 6.

```
<c:iterate select="$d" var="device">
<c:if test="$device/@name">
AddDevice_RefName=<c:get select="$device/@name"/>
</c:if>
AddDevice_RefName=LCase(AddDevice_RefName)
<c:if test="$device/@manufacturer">
AddDevice_Manufacturer=<c:get select="$device/@manufacturer"/>
</c:if>
AddDevice_Manufacturer=LCase(AddDevice_Manufacturer)
<c:if test="$device/@physicalA">
AddDevice_PhysicalAddress=<c:get select="$device/@physicalA"/>
</c:if>
AddDevice_AuxAlreadyAdded=false
AddDevice_AuxName="new device"+AddDevice_RefName
Set AddDevice_auxDR=Project.parent
...

```

This template generates a VBScript macro to create a new device for each of the devices present in the KNX model (obtained from the UML-based component model of Fig. 10). The listing of the obtained macro would look at follows (JET tags have been replaced with KNX device-specific information):

```
'Macro for inserting device 1'
AddDevice_RefName="Universal Dimming Actuator 1-gang REG"
AddDevice_RefName=LCase(AddDevice_RefName)
AddDevice_Manufacturer="Abrecth Jung"
AddDevice_Manufacturer=LCase(AddDevice_Manufacturer)
AddDevice_PhysicalAddress="1.1.2"
AddDevice_AuxAlreadyAdded=false
AddDevice_AuxName="new device"+AddDevice_RefName
Set AddDevice_auxDR=Project.parent
...
'Macro for inserting device 2'
...
'Macro for inserting device n'
...
'More macros for binding, parametrization, etc.'
```

More macros are also obtained for tasks like configuring device parameters and adding binding between devices. Fig. 11 shows a

screenshot of the ETS project generated from the component model of Fig. 10 executing the mentioned macros within the tool. At the top left the devices in the KNX topology are shown. The top right section illustrates the configuration of the DMO-1 device with its communication objects, used in KNX to logically bind different functions through group addresses. The bottom part shows the group addresses as an alternative view of the same project. Each group address links different communication objects of different KNX devices.

This approach for code generation is interesting in the way that it takes advantage of the existing technology tools rather than building a whole set of new tools from scratch.

Without our framework, the starting point for a traditional developer would be the creation of a project by interacting manually with the ETS tool from the outset. With ETS the reuse of partial solutions from application to application is rather constrained. For example, KNX group addresses are fully dependent on the deployment.

4. Traceability between models

In the context of MDD, automated transformation techniques provide capabilities for traceability (Valderas and Pelechano, 2009; Behrens, 2007). Due to the extensive use of transformations (i.e. automated creation of artifacts) used throughout a model driven process it becomes central to be able to understand how and why an artifact was created (Olsen and Oldevik, 2007). The traceability of software artifacts offers much more detailed information on the adaptation of the developed system as well as the implications that any change may have. There are well known key goals of traceability in software development (Kolovos et al., 2006) that can be summarized in the following items:

Fig. 12. A meta-model for traceability.

1. To validate whether the different requirements have been taken into account.
2. To validate whether the obtained implementation complies with requirements and whether they have been satisfied.
3. To identify the impact that any requirements modification may have.

To be able to save and later process the links of the defined traceability it is necessary to have a repository of traces among the different software artifacts generated. In Kolovos et al. (2006) there are two main approaches to deal with traceability in a model-based environment. One is to keep the traceability information embedded in the model itself, as new model elements e.g. as stereotypes or attributes. The other is to keep the traceability information in an external model. This approach has the advantage of keeping the models clean (not polluting them with traceability information) by facilitating loose couplings between the models and the links. The software development framework presented in this paper considers a meta-model dedicated to traceability as described below.

4.1. A meta-model for traceability

There are many traceability meta-models proposed in the literature (Kolovos et al., 2006; Fabro et al., 2006; Aizenbud-Reshef et al., 2006; Dick, 2002; Walderhaug et al., 2006; Melby, 2007; Oldevik and Neple, 2006). The traceability meta-model adopted in our framework is a basic one with enough expressiveness to represent a lot of link types among model elements. This meta-model (see Fig. 12) contains a *TraceLink* pointing to any *ModelElement* via two references: a *source* element and multiple *target* elements. The *ModelElement* is required here because our traceability meta-model

must be able to link to elements of other meta-models. The idea of the *CompositeLink* is to be able to define different levels of granularity to arrange *Links* in others of more complexity, according to the overall purpose that is being referred to. The *linkType* attribute allows developers to categorize the existing relationships and to distinguish on what level of the development process the trace is located.

Traceability models are created during the model transformation process (Walderhaug et al., 2006). The trace between every HA requirement and the corresponding DSL model is created manually by the user. From the DSL through the final generated code, all the traces are obtained automatically as part of each of the model-to-model and model-to-text transformations. In other words, at the same time as transformations are carried out, links between involved elements are recorded in the traceability repository. To achieve this, the transformation rules have been extended in order to record the trace in the defined meta-model.

Fig. 13 includes the extension carried out on the graph of Fig. 9 to include traceability. In this example, only the RHS part of the rule has been modified. It is possible to observe how a new element corresponding to the traceability model appears in the rule. Each *TraceLink* element of the transformation rule collects the necessary information (name, description) for the *linkType* instances of the traceability meta-model. The matches between *source* and *target* are also stored. Thereby, the traces between the DSL and the component-based elements are also recorded.

4.2. A tool for managing traces

Several authors give recommendations about the issues that should be taking into account when designing a tool for managing traceability (Walderhaug et al., 2006; Melby, 2007), such as (1) be able to manage models at different abstraction levels, (2) consider the same meta-model for all the abstraction levels, (3) store the traces in a persistent medium, (4) be able to identify when a trace was created and the location of the referenced element(s), (5) be able to integrate the tool with external applications, and (6) be able to generate the traces both manually and automatically.

With these issues in mind we have developed a tool integrated into the previous framework, which allows developers to generate detailed reports from the trace models. For instance these reports allow HA system developers to analyze how a requirement has been

Fig. 13. Extension example of a graph transformation rule: from the DSL to the component-based level. Black: catalogue view (DSL) instance; red: target (component model) instance; blue: application view (DSL) instance; orange: transformation instance; green: traceability link instance. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of the article.)

Trace Link		Source Link		Target Link		
Name	Description	Metamodel	Model Element	Metamodel	Model Element	Composite?
Light Requirement	Transformation of light requirements corresponding to the fragments of the home automation application (application DSL)					✓
Req3 to DSL	Transformation of requirement 3 to DSL	Requirement	<u>Requirement:hasRequirements</u> - name => On/Off light automatic - presence - description => Automatic Light Switching through a presence detector. - validationProcedure => Walk near the presence detector to verify the correct functioning of the light. - id => REQ3	MetaDomo	<u>MetaDomo:hasFUnitInstance</u> - FUnitName => PIR3-2 - FUnitRef =>	Child of Light Requirement
				MetaDomo	<u>MetaDomo:hasFUnitLink</u> - FUnitName => C5 - canal => true	Child of Light Requirement
				MetaDomo	<u>MetaDomo:hasFUnitInstance</u> - FUnitName => SW7-1 - FUnitRef =>	Child of Light Requirement
				MetaDomo	<u>MetaDomo:hasFUnitLink</u> - FUnitName => L7	Child of Light Requirement
Req2 to DSL	Transformation of requirement 2 to DSL	Requirement	<u>Requirement:hasRequirements</u> - name => Light Dimmer - description => Light intensity regulation through the mechanism, short push for ON/OFF, longer push for Dimmer Up o Down - validationProcedure => Push the mechanism (e.g. pushbutton) short for ON/OFF and longer for regulation in order to verify correct functioning. - id => Req2	MetaDomo	<u>MetaDomo:hasFUnitInstance</u> - FUnitName => PB-1	Child of Light Requirement
				MetaDomo	<u>MetaDomo:hasFUnitLink</u> - FUnitName => C1 - canal => true	Child of Light Requirement
				MetaDomo	<u>MetaDomo:hasFUnitInstance</u> - FUnitName => DMI-1	Child of Light Requirement

Fig. 14. Traceability report created by the tool: from requirements to domain specific descriptions.

taken into account in the process of automatic code generation as well as including information on the solution adopted.

Fig. 14 shows a screen capture of the use of our tool for traceability. For each one of the traces details of name and description are given, as well as the relative information to the source and the target of the traced elements. The first row of the report shows a *CompositeLink* that arranges all the information related to Links involved in the solution using the DSL for the illumination requirement. In this case, Links represent the traces between the requirements number 2 and 3 of Table 1, which have associated as targets the elements of the DSL (functional units and links among them).

As explained below, the traceability report allows the user to distinguish which requirements have been taken into account, which architectural elements give support to each one of them, and lastly, in which physical devices of the platform the requirements have been implemented.

4.3. Benefits derived from traceability

In this subsection, we examine the main benefits that are obtained by using the traceability reports introduced above. These benefits are derived from the possibility of performing the following tasks:

1. *To validate whether all the requirements have been supported.* This validation can be carried out through the traceability reports. In the first place, we can check that all the requirements are represented by means of the model elements obtained according to the DSL. During the process of checking that a requirement has been taken into account, we can filter the traceability report to find those DSL elements that are linked to the analyzed requirement. For example, the requirement number 3 (Table 1) that represents the control of the automatic lighting in relation to the presence of people implies the presence of a functional unit for presence detection (PIR3-2) and of a controller for the power off timer. The analyst can check, according to the traceability report

whether the elements of presence detection and timer are part of the solution of this requirement.

2. *To verify whether the DSL model is compliant with the home automation requirements.* To do this the DSL elements should be validated to see if they fit with the semantics of the relevant requirements. In this case, the traceability report helps the analyst to know which HA requirements have given rise to which elements of the DSL model. To be able to carry out this validation, the analyst should have the templates used in the implementation of the transformation rules perfectly catalogued. The traceability report is required for the verification of the requirements. However, the verification of the correction is not immediate. In the case of the requirement number 3 (Table 1), to assure the correctness of the model implies correctly characterizing the parameterization of the controller for the timer and to verify that its switching off service is correctly linked to the switching services of the lighting elements. These features can be verified more easily with the trace information provided by the traceability tool.
3. *To establish the impact of changing a home automation requirement.* It is usual that the users of the system suggest changes to the requirements throughout the whole development process. The traceability report can help analysts to evaluate the impact of these changes before applying them. The first useful fact for the analyst is to know how many DSL elements, how many architectural components, and how many elements of the specific level of the execution platform should be modified. This impact study is very important because it allows the analyst to obtain an integrated vision of the changes that should be kept in mind when changing a requirement. On the other hand, the analyst can just as easily verify the implications that modifying a requirement of this type has in relation to the rest of the requirements already implanted. For example, they could modify the requirement number 3 (Table 1) to include an additional condition of switching the light on when the light entering from the exterior was insufficient. The inclusion of this modification would

imply the addition of an external light sensor, a controller to compare the light readings with a threshold and the corresponding interconnections with the services of the presence detector and lighting controllers. In view of the integration characteristics of the HA applications, many of these elements can be included in other requirements. For example, the light sensor can be part of the requirement number 5 (control of blinds by lighting level), and the presence detector can be part of the requirements number 8 (change of air conditioning temperature for presence) and 6 (burglar alarm). The traceability report can help the analyst to check which requirement each element is associated to and to verify if the inclusion of new elements or the alteration in their associations can affect the correct implementation of other requirements. In other words, if an external light detector for the automatic power on of lights is required it will be necessary to: check whether this detector is already present to satisfy some other requirement or whether a new element must be added and, to modify the associations of, for example, PIR (presence detection). This PIR can be associated to other DSL elements to implement other requirements. Therefore, when modifying their associations you may be altering the models corresponding to other requirements.

5. Cost model of the proposal

Using catalogues of generic requirements to reuse DSL-based model fragments may have an associated cost and incurs an initial overhead. A model fragment must be built for each generic requirement following the process outlined above. Moreover, building tools for reusing these requirements takes time. The payoff of applying this proposal is realized when the catalogue is used many times. Since catalogues are built with the intention of being reused multiple times, in most of the cases the effort to create and manage the catalogue is paid off. Generally, we can represent the cost of developing a new application from scratch and using the proposed DSL as:

$$\text{Cost (Modeling.App)} = f(n^2 + c) \approx O(n^2),$$

where n represents the number of HA devices and functional units used during modeling and c is the minimum cost of using the tool.

The cost has order n^2 because for each new model element (devices and functional units), the dependencies with each existing element need to be checked. The development of HA applications taking into consideration the reusable requirements catalogue and developed tool for integration can be represented as:

$$\text{Cost (Modeling.App}_R) = C_1 + C_2 + C_3 + f(n + c) \approx O(n + c).$$

The cost $O(n + c)$ is now linear since developers need effort only to integrate the DSL model fragment into the complete model. C_1 represents the extra effort to create and maintain the catalogue. C_2 is the cost of building a more advanced tool infrastructure for managing generic requirements and linking them with DSL model fragments. C_3 is the cost of finding the generic requirements that fits the actual requirements. In our experience, C_1 is not high and tends to zero when the catalogue is stable and robust. C_2 is only a significant overhead initially and is soon paid off. C_3 is a cost that also is also typically low and depends on the previous experience of developers in managing the catalogue. When the catalogue and DSL model fragments are reused multiple times a significant savings is achieved and $\text{Cost (Modeling.App}_R) \approx O(n + c) \ll \text{Cost (Modeling.App)} \approx O(n^2 + c) \approx O(n^2)$.

Table 2
Questions of the survey.

#	Question
Q1	During the third training session I looked into the platform-dependent code in order to customize it very often
Q2	It was very easy to add new requirements
Q3	It was very easy to define a new DSL model fragment from requirements
Q4	It was very easy of to identify, define, connect and classify new requirements
Q5	It was very easy to integrate in a unique model several model fragments from distinct requirements
Q6	I gained time using the new approach
Q7	It was easy to know what a DSL model fragment did
Q8	It was faster to discover a model fragment than building it
Q9	In general terms, I think that reuse of models through generic requirements is improved
Q10	When I start a new HA project, how would I proceed?

6. Evaluation of the proposal

To measure the success of reusing model fragments through generic requirements we conducted a survey among HA developers. The survey was set up according to [Pfleeger and Kitchenham \(2001\)](#). The first step was to set the survey objective(s). The objective of our evaluation was to provide an answer to the following research question:

Is the use of the framework really effective for reusing DSL-based models through generic requirements?

To reason about the effectiveness of reuse, we identified a number of questions (see [Table 2](#)) that gives to a lesser or greater extent the success in reusing models. These questions relate directly to the survey objective and consider the availability of the previous methods, languages, and tools. Taking into account ([Pfleeger and Kitchenham, 2001](#)), the questions have the following properties: they are neutral, in other words, we have avoided the use of wording that could influence the answer; they adequately cover the topic; the order of questions was independent so that the answer to one does not influence the response to the next; they represent unbiased and mutually exclusive response categories. After pre-testing the questionnaire (i.e. reliability, understandability, validity, etc.) it was conducted. Before undertaking any detailed analysis, responses were vetted for consistency and completeness as ([Pfleeger and Kitchenham, 2001](#)) states. In total we invited 12 experts in HA development to participate in this survey. The experts were located in companies engaged in the development of HA applications. They were asked to participate in the study and a random selection was made from those who were willing to participate. Therefore, we knew for sure that all of them had a background in developing HA systems. All participants responded and we got 11 meaningful results, giving our survey an effective response rate of 92%. With that small population it was not used any form of sampling.

Developers received the questionnaire with an explanation of the purpose of the survey. We imposed no time limit in filling out the questionnaire. Since the mentioned experts were neither involved in the design of the DSL nor in the methodology there were no risks of bias towards a positive evaluation. This assures us that subjects got the chance to reflect both positive and negative aspects of the proposal. We offered participants three training sessions before beginning the evaluation. The first involved training in the use of the defined DSL and tools (3 h). The second session provided training in the use of the requirements catalogue and the previously detailed procedures for managing the catalogue (2 h). The third session was dedicated to solve a medium-size HA example and then to modify it with new capabilities (4 h). This last session involved

Table 3
Outcomes of the developed survey.

#	Strongly disagree	Disagree	Neutral	Agree	Strongly agree
Q1	3	5	2	1	0
Q2	0	1	2	5	3
Q3	0	1	3	6	1
Q4	0	1	2	5	3
Q5	1	3	4	2	1
Q6	0	1	2	4	4
Q7	0	2	3	4	2
Q8	0	1	2	6	2
Q9	0	1	2	5	3
#	Use the catalogue and DSL fragments	Start a new DSL model from scratch	Copy-modify previous models	Use platform-dependent tools	
Q10	6	3	2	0	

scenarios where all the situations were simulated (new categories, new devices, new requirements, new requirements relationships, etc.).

As Table 3 shows, the outcomes of the study indicate that in the given survey the framework helped to improve reuse, and therefore, to reduce time and costs. The distribution of the answers to question number 1 demonstrates the confidence that commercial platform-dependent developers have in the tool. It also shows the need for long-term experiences that provide greater confidence in the proposed approach. The responses to question 2 show the difficulties encountered by some users in the management of the requirements catalogue. We think that most of these difficulties were due to the fact that these users were not familiar with the way in which the requirements were organized. Again, deficiencies related to this aspect would diminish with the continued use of the catalogue. Questions 3 and 7 demonstrate the capabilities that our tool provides for the integration and interpretation of the DSL fragments in the catalogue, one of the main goals of our proposal. Question number 5 highlights the extra effort needed to integrate model fragments. The answers to questions 6 and 8 show that time is gained using our tool. We believe that performance improvements can be even more significant as both the tool and the developers' training evolve. Question number 10 is relevant in this study because it reflects the improvement in the development process. Moreover, question number 9 summarizes the favorable view of developers about the proposal regarding requirements and DSL fragments reuse.

Table 4 summarizes a comparison of our tool with two HA well known commercial tools based in our experience. As it can be seen, the results highlight the interest to continue working on the research line proposed in this paper.

7. Conclusions and future work

In this work we have introduced a framework for developing HA systems by following a model driven approach. The development of this kind of systems is a time-consuming endeavor that could only be afforded by experienced developers with programming and execution-platform expertise. Our framework allows low-experienced developers to create HA system descriptions by means of a domain specific language and a set of transformations to generate executable code. These features enable the development of HA applications with improved quality both of the process and of the software artifacts obtained.

Our approach makes it possible to deal with the reuse of model fragments from the requirements elicitation phase. It provides a requirements catalogue that can be reused from system to system development. This catalogue has served as a basis to develop an extensible approach that allows developers to reuse DSL-model fragments and hence to incorporate as a partial solution that will be integrated into complete DSL models. This helps modelers to improve the quality of their models and avoid the necessity of modeling recurrent requirements. A set of experiments to help provide an empirical evaluation of the proposal has been given. Thus, the survey demonstrates that the reuse of DSL-based models through generic requirements is really effective. The most encouraging lesson from our work is the opportunity to increase the level of model reuse through the combination of DSLs and generic requirements. It is the synergy between the involved elements in the process which is likely to bring the most significant benefits of reuse in MDD. The factor of usability of the DSL graphical syntax has also been tested with very promising results as detailed in Jiménez et al. (2009).

Table 4
Comparison of features of the developed tool and the two leading HA technology-dependent tools.

Feature	Developed tool	ETS (KNX/EIB)	Lonmaker (Lonworks)
Concepts used by the tool user	Exclusively HA domain concepts	HA domain + platform dependent concepts	HA domain + platform dependent concepts
Requirement support	Yes	No	No
Graphical syntax for HA concepts	Yes	No	Partially
Platform independent modeling	Yes	No	No
Design reuse	Very easy	Hard	Medium
Capability of integrating different HA technologies	Yes (through future extensions in transformations)	No	No
Possibility of integrating other domains (wireless sensor networks, computer vision, robotics, etc.)	Yes (by means of the platform independent component model)	No	No
Traceability support	Yes	No	No

Our work is now focused on the consideration of other execution platforms and the interoperability of our framework with other tools (such as an animator to validate the models before final deployment). In addition, work is currently underway to improve the traceability tool to facilitate, for instance, advanced filtering features, statistical information of elements generated in each level, analysis of orphan elements (a typical use of this is to find elements that are not required by the system, e.g. a feature that was not described in the requirements) and, lastly, the integration of the tool in environments other than Eclipse.

Also of interest is the consideration of the BCU SDK tools from the University of Vienna (Neugschwandtner et al., 2005) which offer an alternative way to tackle the code generation for the KNX/EIB platform. These free tools could facilitate the code generation as they allow writing customized code for OEM KNX devices. Thus the need of cataloging commercial devices by functionality could be avoided.

The elements of the domain specific language can be extended and refined to consider more specific elements. We also want to extend the language in order to facilitate the modeling of new devices from scratch. All this information will be very valuable to extend and tune future versions of the framework to developer requirements.

Acknowledgements

This work was partially supported by the Spanish CICYT project EXPLORE (TIN2009-08572). We are also grateful for highly constructive engagement by the reviewers and editors.

References

- Aizenbud-Reshef, N., Nolan, B.T., Rubin, J., Shaham-Gafni, Y., 2006. Model traceability. *IBM Systems Journal* 45 (3), 515–526.
- Behrens, T., 2007. Never Without a Trace: Practical Advice on Implementing Traceability. <http://www.ibm.com/developerworks/rational/library/feb07/behrens>.
- Biermann, E., Ehrig, K., Ermel, C., Köhler, C., Kuhns, G., Taentzer, G., 2006. Tiger EMF Model Transformation Framework (EMT). <http://tfs.cs.tu-berlin.de/emftrans/papers/userdoc.pdf>.
- Bühne, S., Halmans, G., Lauenroth, K., Pohl, K., 2006. Scenario-based application requirements engineering. In: *Software Product Lines: Research Issues in Engineering and Management*. Springer, Heidelberg.
- Chana, D., Campoa, E., Estevea, D., Fourniols, J.Y., 2009. Smart homes: current features and future perspectives. *Maturitas* 64 (2), 90–97.
- Clements, P., Northrop, P., 2002. *Software Product Lines: Practices and Patterns*. Boston, Addison-Wesley.
- Dick, J., 2002. Rich traceability. In: *Proceedings of Automated Software Engineering*, CA, USA.
- DSM Forum, 2010. <http://www.dsmforum.org>.
- Eclipse Consortium, 2010. Java Emitter Templates (JET). <http://www.eclipse.org/modeling/m2t/?project=jt>.
- Fabro, D., Bezivin, J., Valduriez, P., 2006. Weaving models with the eclipse AMW plugin. In: *Eclipse Modeling Symposium, Eclipse Summit Europe*, Esslingen, Germany.
- Hermans, F., Pinzger, M., van Deursen, A., 2009. Domain-specific languages in practice: a user study on the success factors. *Lecture Notes in Computer Science* 5795, 423–437.
- ITTools, 2010. IT Gesellschaft für Informationstechnik mbH. <http://www.it-gmbh.de/en/products/ittools.htm>.
- Jacob, L., Reed, K., 2000. Requirements classification and reuse: crossing domain boundaries. 6th International Conference on Software Reuse: Advances in Software Reuse, Vienna, Austria. *Lecture Notes in Computer Science* 1844, 190–210.
- Jammes, F., Mensch, A., Smit, H., 2005. Service-oriented device communications using the devices profile for web services. In: *Proceedings of the 3rd International Workshop on Middleware for Pervasive and Ad-hoc Computing*, CA, USA.
- Jimenez, M., 2009. Development of HA Applications Using a Model Driven Approach. Ph.D. Thesis. Technical University of Cartagena, Spain, (March). <http://hdl.handle.net/10317/846>.
- Jiménez, M., Rosique, F., Sánchez, P., Álvarez, B., Iborra, A., 2009. Habitation: a domain-specific language for home automation. *IEEE Software* 26 (4), 33–38.
- Kärnä, J., Tolvanen, J.P., Kelly, S., 2010. Evaluating the Use of Domain-specific Modeling in Practice. <http://www.dsmforum.org/events/DSM09/Papers/Karna.pdf>.
- Kelly, S., Tolvanen, J.P., 2008. Domain-specific Modeling: Enabling Full Code Generation. John Wiley & Sons.
- Kolovos, D., Paige, R., Polack, F., 2006. On-demand merging of traceability links with models. In: *Proceedings of the 2nd EC-MDA Workshop on Traceability*, Bilbao, Spain.
- Lam, W., 1998. A case-study of requirements reuse through product families. *Annals of Software Engineering* 5, 253–277.
- Lam, W.J., McDermid, J.A., Vickers, A.J., 1997. Ten steps towards systematic requirements reuse. *Journal of Requirements Engineering* 2, 102–113.
- Melby, S., 2007. Traceability in Model Driven Engineering. Master Thesis. University of Oslo Norway. <http://urn.nb.no/URN:NBN:no-18721>.
- Mens, T., van Gorp, P., 2006. A taxonomy of model transformation. *Electronic Notes in Theoretical Computer Science* 152, 125–142.
- Mernik, M., Heering, J.A., Sloane, M., 2005. When and how to develop domain-specific languages. *ACM Computing Surveys* 37 (4), 316–344.
- Miori, V., Tarrini, L., Manca, M., Tolomei, G., 2006. An open standard solution for domestic interoperability. *IEEE Transactions on Consumer Electronics* 52 (1), 97–103.
- Molina, F., Toval, A., 2009. Integrating usability requirements that can be evaluated in design time into model driven engineering of web information systems. *Advances in Engineering Software* 40, 1306–1317.
- Muñoz, J., Pelechano, V., Cetina, C., 2006. Implementing a pervasive meetings room: a model driven approach. In: *Proceeding of the 3rd International Workshop on Ubiquitous Computing*, Paphos, Cyprus.
- Nain, G., Dauber, E., Barais, O., Jézéquel, J.M., 2008. Using mde to build aschizofrenic middleware for home/building automation. In: *ServiceWave'08: Networked European Software & Services Initiative (NESSI) Conference*, Madrid, Spain.
- Neugschwandtner, G., Kastner, W., Kogler, M., 2005. Programming fieldbus nodes: a RAD approach to customizable applications. In: *10th IEEE Conference on Emerging Technologies and Factory Automation (ETFA'05)*, Catania, Italy.
- Oldevik, J., Neple, T., 2006. Traceability in model to text transformations. In: *2nd European Conference on Model-Driven Architecture Foundations and Applications*, Bilbao, Spain.
- Olsen, G.K., Oldevik, J., 2007. Scenarios of traceability in model to text transformations. In: *3rd European Conference on Model-Driven Architecture Foundations and Applications*, Haifa, Israel.
- Pfleeger, S., Kitchenham, B., 2001. Principles of survey research. *ACM SIGSOFT Software Engineering Notes* 26, 16–18.
- Ramesh, B., Jarke, M., 2001. Toward reference models for requirements traceability. *IEEE Transactions on Software Engineering* 27 (1), 58–93.
- Rozenberg, G., 1997. *Handbook of Graph Grammars and Computing by Graph Transformation*. World Scientific, Netherlands.
- Selic, B., 2003. The pragmatics of model-driven development. *IEEE Software* 20 (5), 19–25.
- Valderas, P., Pelechano, V., 2009. Introducing requirements traceability support in model-driven development of web applications. *Information and Software Technology* 51 (4), 749–768.
- Vicente-Chicote, C., Moros, B., Toval, A., 2007. Remm-studio: an integrated model-driven environment for requirements specification, validation and formatting. *Journal of Object Technology* 6 (9), 437–454.
- Voelter, M., Groher, I., 2007. Product line implementation using aspect-oriented and model-driven software development. In: *Proceedings of the 11th International Software Product Line Conference*, Kyoto, Japan.
- von Kneten, A., Paech, B., Kiedaisch, F., Houdek, F., 2002. Systematic requirements recycling through abstraction and traceability. In: *10th International Requirements Engineering Conference (RE'02)*, Essen, Germany.
- Walderhaug, S., Johansen, U., Stav, E., Aagedal, J., 2006. Towards a generic solution for traceability in MDD. In: *Proceedings of the 2nd EC-MDA Workshop on Traceability*, Bilbao, Spain.

Pedro Sánchez is an associate professor of computer science at the Technical University of Cartagena and a member of the university's DSIE (Division of Systems and Electronic Engineering) research group. His research interests include model-driven engineering for real-time systems. Sánchez has a PhD in computer science from the Technical University of Valencia.

Manuel Jiménez is an associate professor of industrial electronics at the Technical University of Cartagena and is a member of the university's DSIE (Division of Systems and Electronic Engineering) research group. His research interests include electronics and model driven engineering for reactive systems. Jiménez has a PhD in computer science from the Technical University of Cartagena.

Francisca Rosique is an assistant professor and a PhD student in computer science at the Technical University of Cartagena and a member of the university's DSIE (Division of Systems and Electronic Engineering) research group. Her research interests include model-driven engineering and home automation systems. Rosique has a master's in telecommunication engineering from the Technical University of Cartagena.

Bárbara Álvarez is a full professor in computer science at the Technical University of Cartagena and a member of the university's DSIE (Division of Systems and Electronic Engineering) research group. Her research interests include real-time systems and software architectures for teleoperation. Alvarez has a PhD in telecommunication engineering from the Technical University of Madrid.

Andrés Iborra is full professor and head of the Electronics Technology Department at the Technical University of Cartagena and a member of the university's DSIE (Division of Systems and Electronic Engineering) research group. His research interests include computer vision and robotics. Iborra has a PhD in industrial engineering from the Technical University of Madrid.

Impact Factor Trend Graph: JOURNAL OF SYSTEMS AND SOFTWARE

Click on the "Return to Journal" button to view the full journal information.

***Impact Factor -- see below for calculations**

The journal impact factor is a measure of the frequency with which the "average article" in a journal has been cited in a particular year. The impact factor will help you evaluate a journal's relative importance, especially when you compare it to others in the same field. For more bibliometric data and information on this and other journal titles click on the "Return to Journal" button.

NOTE: Title changes and coverage changes may result in no impact factor for one or more years in the above graph.

2010 Impact Factor

Cites in 2010 to articles published in:	2009 = 132	Number of articles published in:	2009 = 167
	2008 = 292		2008 = 161
	Sum: 424		Sum: 328

Calculation: $\frac{\text{Cites to recent articles}}{\text{Number of recent articles}} = \frac{424}{328} = \mathbf{1.293}$

2009 Impact Factor

Cites in 2009 to articles published in:	2008 = 158	Number of articles published in:	2008 = 161
	2007 = 260		2007 = 151
	Sum: 418		Sum: 312

Calculation: $\frac{\text{Cites to recent articles}}{\text{Number of recent articles}} = \frac{418}{312} = \mathbf{1.340}$

2008 Impact Factor

Cites in 2008 to articles published in:	2007 = 183	Number of articles published in:	2007 = 151
	2006 = 172		2006 = 135

Sum: 355
Calculation: $\frac{\text{Cites to recent articles}}{\text{Number of recent articles}} = \frac{355}{286} = \mathbf{1.241}$

Sum: 286

2007 Impact Factor

Cites in 2007 to articles published in: 2006 = 79 Number of articles published in: 2006 = 135
2005 = 128 2005 = 124
Sum: 207 Sum: 259

Calculation: $\frac{\text{Cites to recent articles}}{\text{Number of recent articles}} = \frac{207}{259} = \mathbf{0.799}$

2006 Impact Factor

Cites in 2006 to articles published in: 2005 = 65 Number of articles published in: 2005 = 124
2004 = 103 2004 = 160
Sum: 168 Sum: 284

Calculation: $\frac{\text{Cites to recent articles}}{\text{Number of recent articles}} = \frac{168}{284} = \mathbf{0.592}$

[Acceptable Use Policy](#)
Copyright © 2012 [Thomson Reuters](#).

THOMSON REUTERS

Published by Thomson Reuters

Rank in Category: JOURNAL OF SYSTEMS AND SOFTWARE**Journal Ranking**

For **2010**, the journal **JOURNAL OF SYSTEMS AND SOFTWARE** has an Impact Factor of **1.293**.

This table shows the ranking of this journal in its subject categories based on Impact Factor.

Category Name	Total Journals in Category	Journal Rank in Category	Quartile in Category
COMPUTER SCIENCE, SOFTWARE ENGINEERING	99	34	Q2
COMPUTER SCIENCE, THEORY & METHODS	97	35	Q2

Category Box Plot

For **2010**, the journal **JOURNAL OF SYSTEMS AND SOFTWARE** has an Impact Factor of **1.293**.

This is a box plot of the subject category or categories to which the journal has been assigned. It provides information about the distribution of journals based on Impact Factor values. It shows median, 25th and 75th percentiles, and the extreme values of the distribution.