The Journal of Systems and Software xxx (2011) xxx–xxx

Contents lists available at ScienceDirect

The Journal of Systems and Software

journal homepage: www.elsevier.com/locate/jss

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 xxx

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.

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 standards 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).

0164-1212/$ – see front matter © 2011 Elsevier Inc. All rights reserved.
doi:10.1016/j.jss.2011.01.052

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 Knethen 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

Please cite this article in press as: Sánchez, P., et al., A framework for developing home automation systems: From requirements to code. J. Syst. Software (2011), doi:10.1016/j.jss.2011.01.052

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

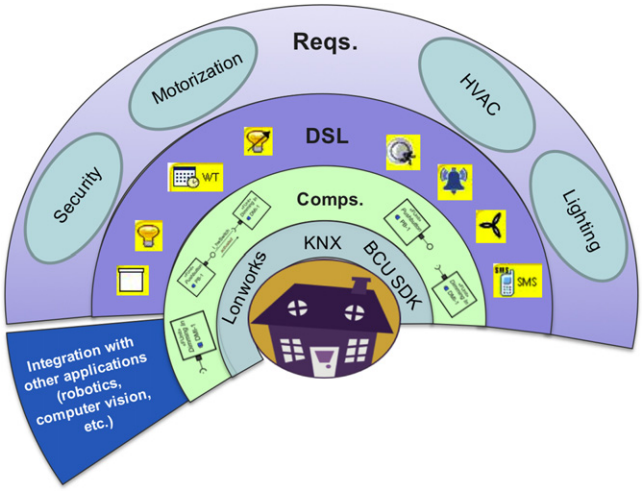


Fig. 1. Overview of the framework.

at (Jimenej, 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

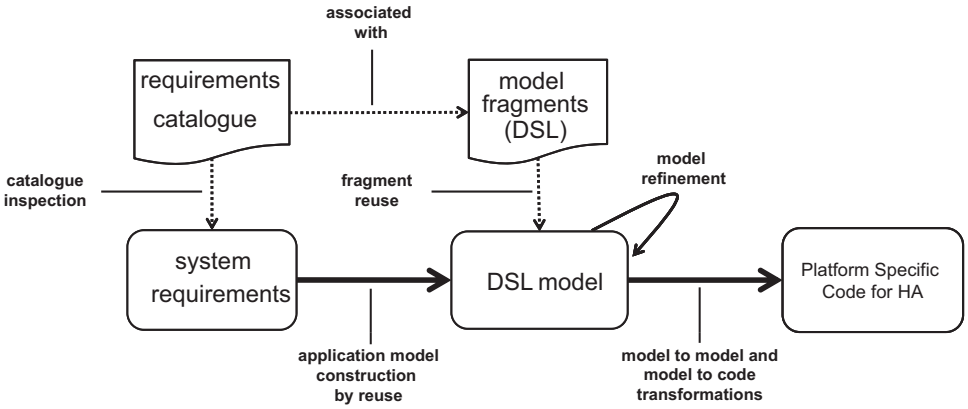


Fig. 2. A scheme of the methodology.

Please cite this article in press as: Sánchez, P., et al., A framework for developing home automation systems: From requirements to code. J. Syst. Software (2011), doi:10.1016/j.jss.2011.01.052

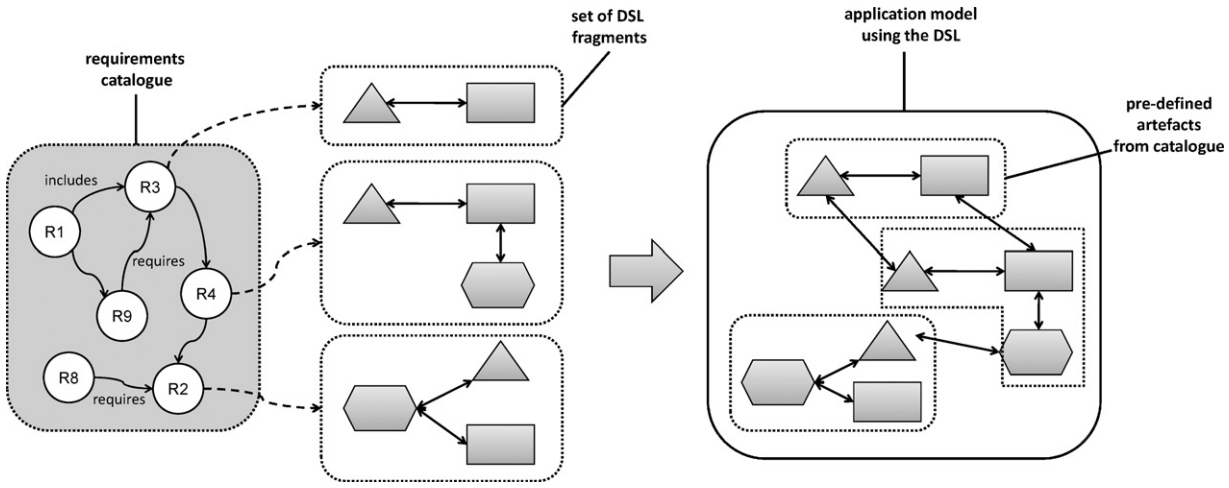


Fig. 3. Overview of the approach: from generic requirements to integrated models.

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 (*Requirement*) 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.

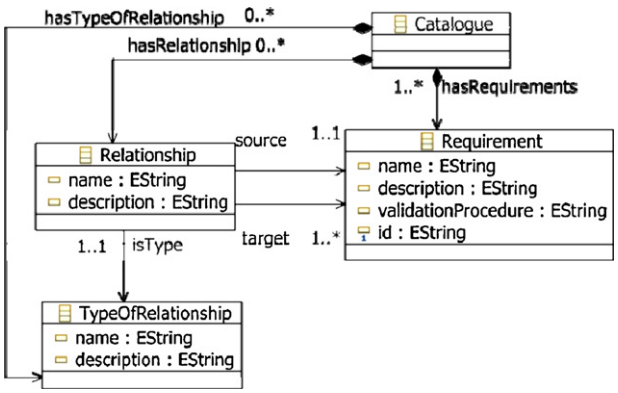


Fig. 4. Basic requirements meta-model.

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 (Jimenej, 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

Please cite this article in press as: Sánchez, P., et al., A framework for developing home automation systems: From requirements to code. J. Syst. Software (2011), doi:10.1016/j.jss.2011.01.052

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 $\pm 4\,^{\circ}C$ 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*".

Please cite this article in press as: Sánchez, P., et al., A framework for developing home automation systems: From requirements to code. J. Syst. Software (2011), doi:10.1016/j.jss.2011.01.052

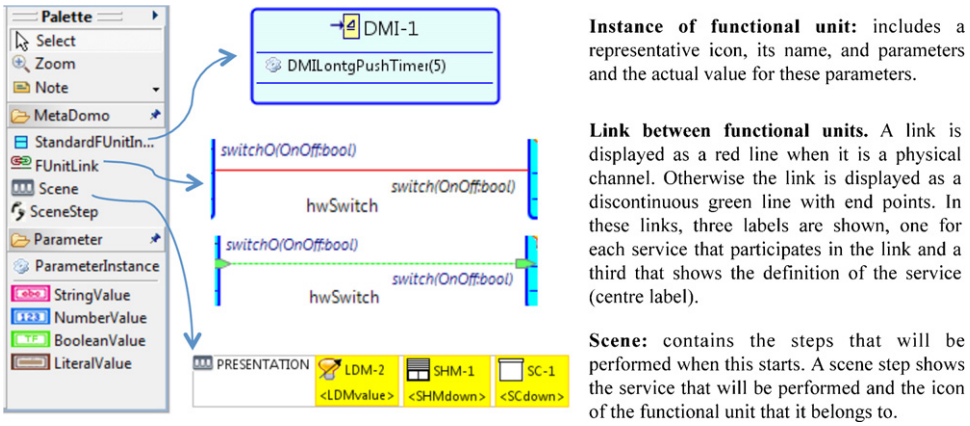


Fig. 5. Graphical representation and description of some of the elements used in the graphical language.

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

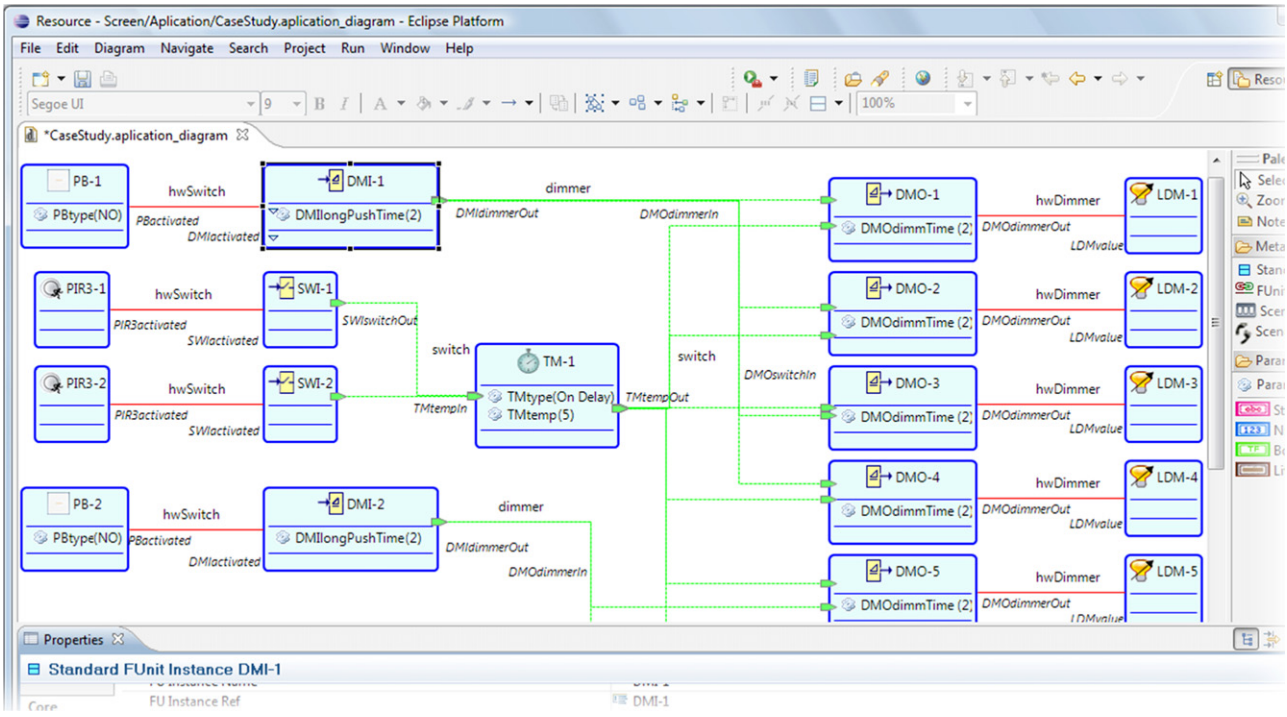


Fig. 6. A screenshot of the tool including the DSL model for a lighting application with comfort and energy saving functions.

Please cite this article in press as: Sánchez, P., et al., A framework for developing home automation systems: From requirements to code. J. Syst. Software (2011), doi:10.1016/j.jss.2011.01.052

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

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

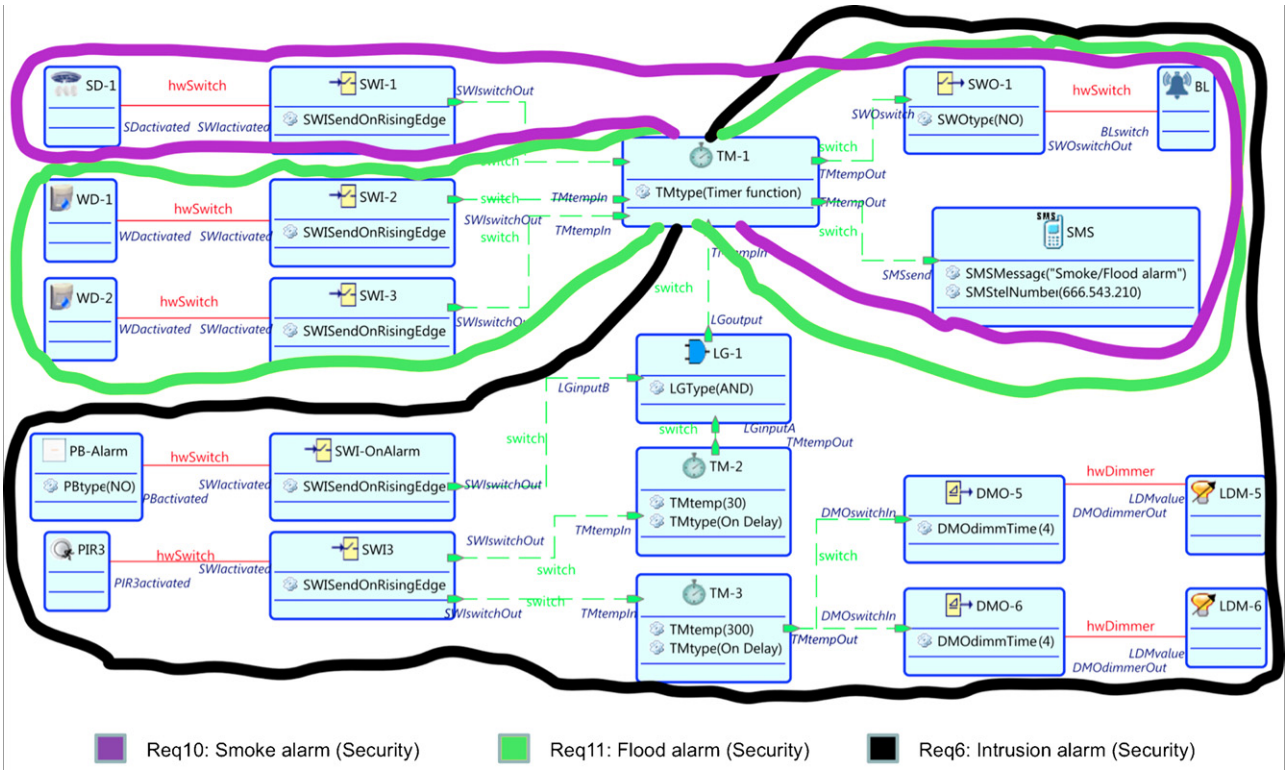


Fig. 8. DSL fragments for various requirements needed within the previous application.

Please cite this article in press as: Sánchez, P., et al., A framework for developing home automation systems: From requirements to code. J. Syst. Software (2011), doi:10.1016/j.jss.2011.01.052

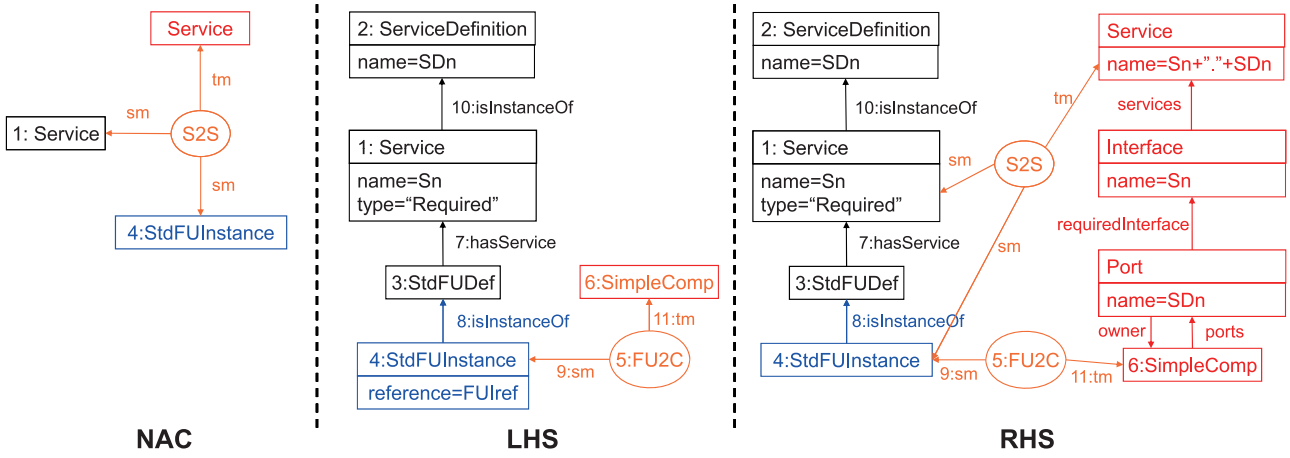


Fig. 9. Example of graph transformation rule: from DSL to component model level. Black: catalogue view (DSL) instance; red: target (component model) instance; blue: application view (DSL) instance; orange: transformation instance. (For interpretation of the references to color in this figure legend,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

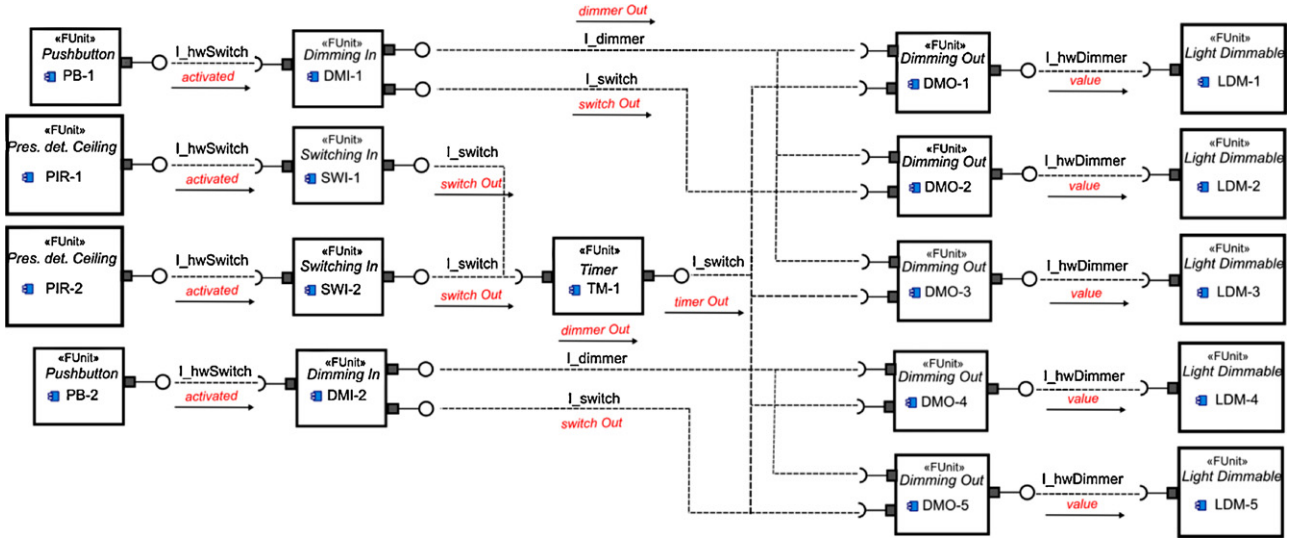


Fig. 10. UML-based component model obtained after applying the graph transformation rules to the DSL example of Fig. 6.

Please cite this article in press as: Sánchez, P., et al., A framework for developing home automation systems: From requirements to code. J. Syst. Software (2011), doi:10.1016/j.jss.2011.01.052

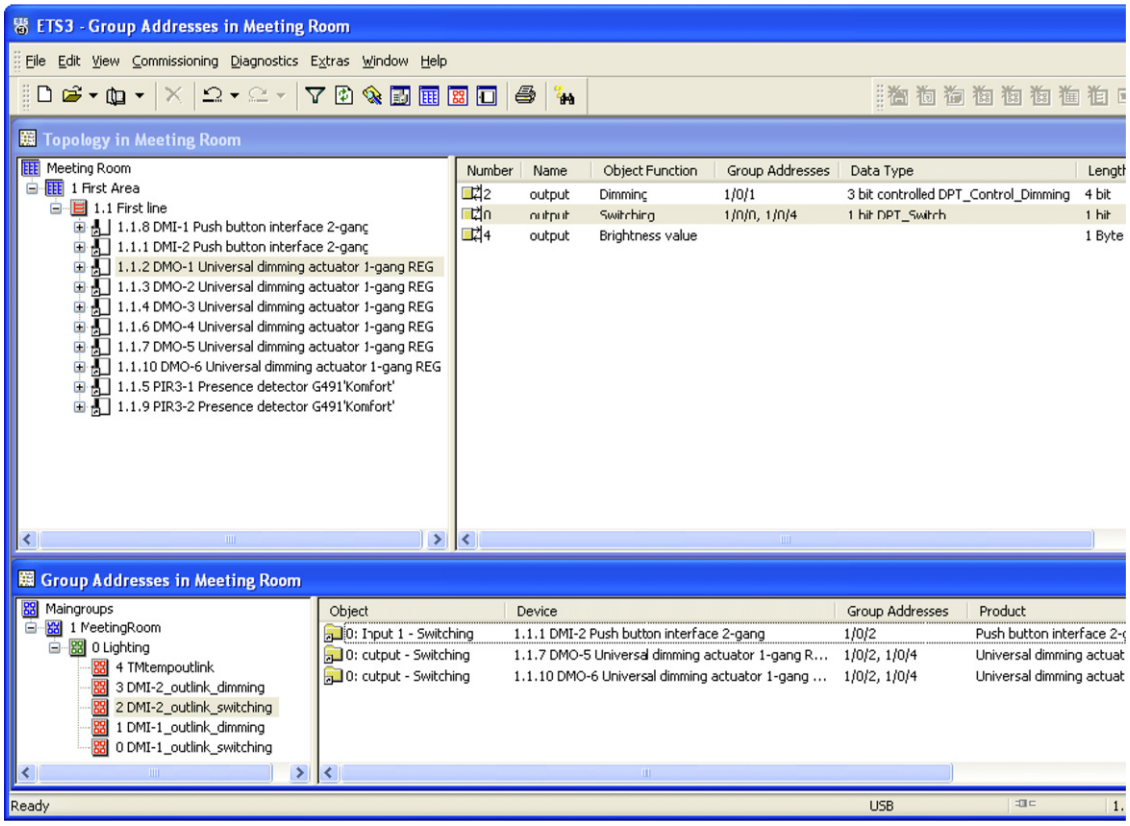


Fig. 11. ETS Project obtained after execution of the ITTools macros for the example of Fig. 6.

```
<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:

Please cite this article in press as: Sánchez, P., et al., A framework for developing home automation systems: From requirements to code. J. Syst. Software (2011), doi:10.1016/j.jss.2011.01.052

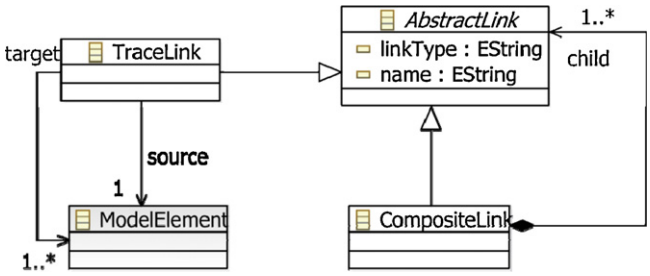


Fig. 12. A meta-model for traceability.

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

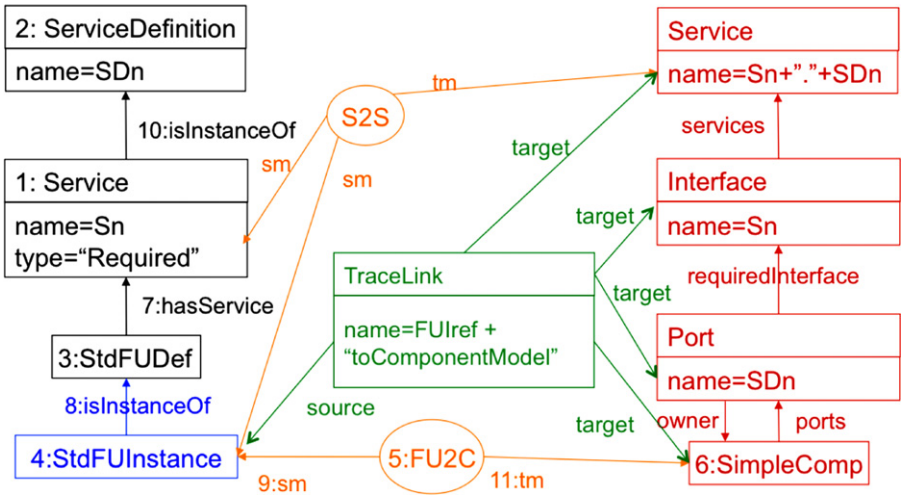
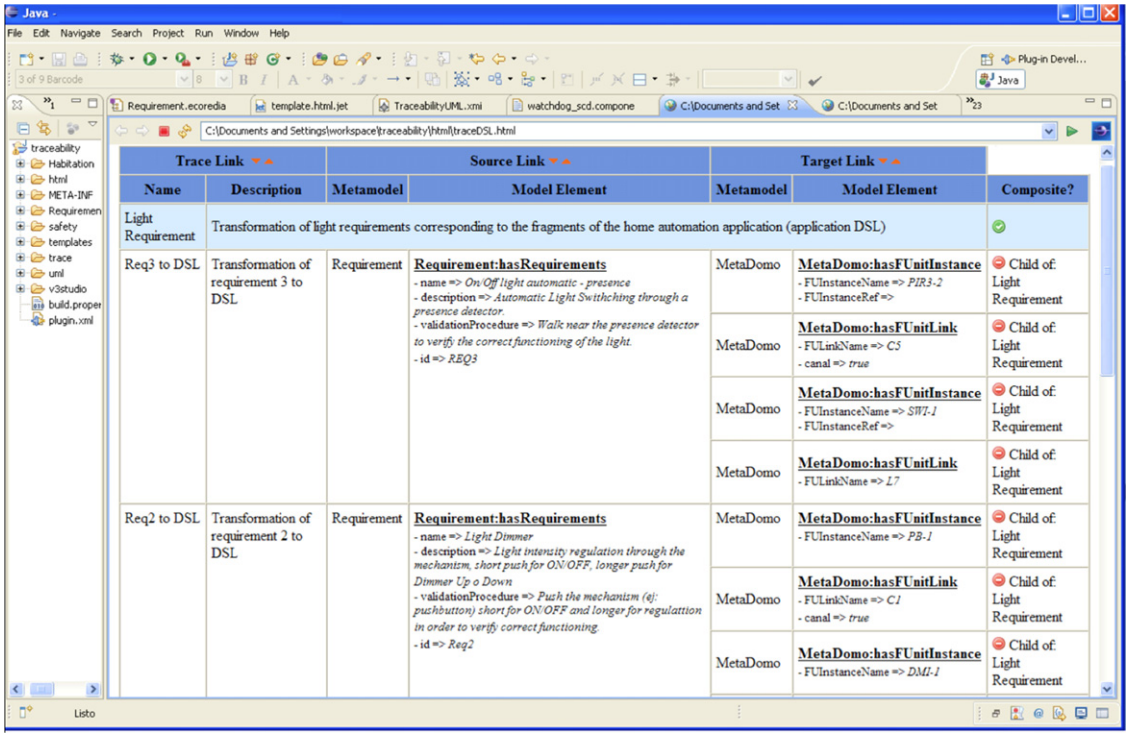


Fig. 13. Extension example of a graph transformation rule: from the DSL to the component-based level. Black: catalogue view (DSL) instance; red: target (component model) instance; blue: application view (DSL) instance; orange: transformation instance; green: traceability link instance. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of the article.)

Please cite this article in press as: Sánchez, P., et al., A framework for developing home automation systems: From requirements to code. J. Syst. Software (2011), doi:10.1016/j.jss.2011.01.052



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	**Requirement:hasRequirements** - name => *On/Off light automatic - presence* - description => *Automatic Light Swithching through a presence detector.* - validationProcedure => *Walk near the presence detector to verify the correct functioning of the light.* - id => *REQ3*	MetaDomo	**MetaDomo:hasFUnitInstance** - FUInstanceName => *PIR3-2* - FUInstanceRef =>	Child of: Light Requirement
				MetaDomo	**MetaDomo:hasFUnitLink** - FULinkName => *C5* - canal => *true*	Child of: Light Requirement
				MetaDomo	**MetaDomo:hasFUnitInstance** - FUInstanceName => *SWI-1* - FUInstanceRef =>	Child of: Light Requirement
				MetaDomo	**MetaDomo:hasFUnitLink** - FULinkName => *L7*	Child of: Light Requirement
Req2 to DSL	Transformation of requirement 2 to DSL	Requirement	**Requirement:hasRequirements** - 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): pushbutton) short for ON/OFF and longer for regulattion in order to verify correct functioning.* - id => *Req2*	MetaDomo	**MetaDomo:hasFUnitInstance** - FUInstanceName => *PB-1*	Child of: Light Requirement
				MetaDomo	**MetaDomo:hasFUnitLink** - FULinkName => *C1* - canal => *true*	Child of: Light Requirement
				MetaDomo	**MetaDomo:hasFUnitInstance** - FUInstanceName => *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

Please cite this article in press as: Sánchez, P., et al., A framework for developing home automation systems: From requirements to code. J. Syst. Software (2011), doi:10.1016/j.jss.2011.01.052

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:

$$\text{Cost}\ \ (\text{Modeling\_App}) = f(n^2 + c) \approx O(n^2),$$

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:

$$\text{Cost}\ \ (\text{Modeling\_App}_R) = C_1 + C_2 + C_3 + f(n + c) \approx O(n + c).$$

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 $\text{Cost}\ (\text{Modeling\_App}_R) \approx O(n+c) \ll \text{Cost}\ (\text{Modeling\_App}) \approx O(n^2+c) \approx O(n^2)$.

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

Please cite this article in press as: Sánchez, P., et al., A framework for developing home automation systems: From requirements to code. J. Syst. Software (2011), doi:10.1016/j.jss.2011.01.052

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

Please cite this article in press as: Sánchez, P., et al., A framework for developing home automation systems: From requirements to code. J. Syst. Software (2011), doi:10.1016/j.jss.2011.01.052

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, M., Campoa, E., Estevea, D., Fourniols, J.Y., 2009. Smart homes: current features and future perspectives. Maturitas 64 (2), 90–97.
Clements, P., Northrop, L., 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=jet.
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 GesellschaftfürInformationstechnikmbH., 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 Reusability, 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 domotic 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 Knethen, 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.

Please cite this article in press as: Sánchez, P., et al., A framework for developing home automation systems: From requirements to code. J. Syst. Software (2011), doi:10.1016/j.jss.2011.01.052