LESSONS LEARNED IN APPLYING MDE TO THE DEVELOPMENT OF HOME AUTOMATION SYSTEMS

Francisca Rosique, Pedro Sánchez, Manuel Jiménez and Diego Alonso
DSIE Research Group, Technical University of Cartagena, Campus Muralla del Mar s/n, E-30202, Cartagena, Spain

Keywords: Home Automation, Model Driven Engineering, Model Transformations, Domain Specific Languages.

Abstract: Home Automation (HA) systems represent a domain of interest to evaluate the benefits and difficulties of adopting the well known Model Driven Engineering (MDE) approach. This is due to the existence of determining factors in the development of such systems that makes MDE applicable with some considerations. This article presents the lessons learned after the definition of a methodology and the implementation of a set of tools to support the MDE-base development of HA systems. In particular, the definition of a Domain Specific Language has made possible the generation of code although we have identified some peculiarities and differences from a classical MDE perspective. These results can be extrapolated to other domains with similar characteristic.

1 INTRODUCTION

Home Automation (HA) systems have emerged as a field of great interest in the engineering field given the current demand for information systems in society. One of the main problems of HA systems 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 examples worth mentioning are the KNX (ISO/IEC14543-3-X)) (Karlheinz, 2009), Lonworks (ISO/IEC 14908)(Echelon, 2009) and X10(Technica, 2005) technologies. Furthermore, as stated in (Miori, 2006), 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. Thus, the development of these systems requires developers that have a very high degree of specialization in the used technology. But all these tools are, unfortunately, incompatible among them, and thus applications cannot be reused in HA systems that use different technologies.

The *Model-Driven Engineering* (MDE) approach (Selic, 2003) presentsa very promising alternative to solve the problems of the current development techniques for HA systems, as mentioned before. MDE provides a theoretical and technological framework for the use and management of models in software development. The work presented in this paper describeshow MDE has been used to develop a framework for HA applications that(i) providesan approach more in keeping with the principles of Software Engineering, and (ii) considers the product life-cycle.

The article (Sanchez, 2011)presentsour integrated framework that allows the definition of HA systems at different levels of abstraction, from requirements to code, as can be seen in Figure 1.

These levels of abstraction have been organized according to the Object Management Group's *Model-Driven Architecture* (MDA) (Mellor, 2004) initiative, into a *Computation-Independent Model* (CIM), a *Platform-Independent Model* (PIM), and several *Platform-Specific Models* (PSM).

The steps taken to obtain this framework were the following:

- A preliminary analysis of the current state of HA systems (domain analysis) and identification of the HA domain concepts was carried out.

- Acatalog of reusable HA requirements was created.
- A definition of the various software artefacts (functional units, components, etc.) that can be assembled and configured was done.
- Languages and tools for both the representation of the specific requirements of each system and the automatic transformation towards lower levels of the MDA approach, which are closer to the platform, were developed.
- A tool for traceability management through was developed and integrated in the framework.

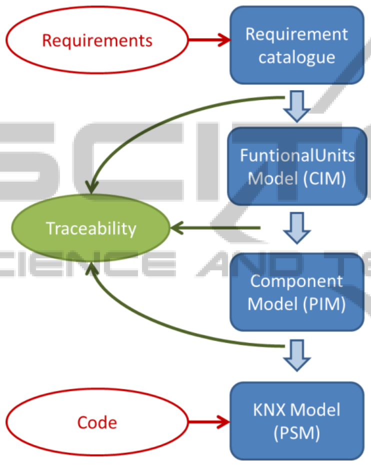

Figure 1: Framework for home automation.

This paper presents the lessons learned from the decisions taken, the difficulties encountered and the conclusions reached over the creation of a HA development framework with the aforementioned characteristics, that is from requirements to code. Section 2 presents the main lessons learned during its development, and finally section 3 outlines the conclusions.

2 LESSONS LEARNED

In the previous section we have summarized the main characteristics of the developed framework, the tools used, and the steps that led us to its development. This section describes the main lessons learned from this work. These lessons seek to extract, from the achievements and constraints, conclusions and knowledge that can help other developers when facing the development of a similar framework in similar domains.

Lesson 1: About the Importance of Platform Independence

Following the MDA guidelines, it is essential to keep platform independence in the early stages of the development process. In the HA domain, the differences between platforms, technologies, standards and manufacturers are significant. Most of the time the compatibility is not guaranteed, so the early selection of physical devices (switches, controllers, alarms, etc.) and an implementation technology (EIB, for instance) is usually inevitable. This commitment to a specific platform makes it difficult to carry to term an MDE approach.

The domain analysis performed previously to the framework development demonstrates that there are functional elements that appear in all HA technologies and standards, although these technologies differ in their architecture, protocols or available modules. We have called these elements "*functional units*", and they form the core of the approach, since we model HA systems according to the functional units needed. In the later stages, and depending on the selected technology, specific devices that provide the functionality will be selected, generating the corresponding code after a model transformation step. In short, platform independence has been achieved thanks to the study of the domain and the consideration of the functional units as basic elements to describe the HA systems.

Lesson 2: Reuse is Challenging

As Krueger states in (Krueger, 1992), "*software reuse is the process of creating software systems from existing software rather than building software systems from scratch*". In this context, abstraction plays a key role, reducing the time and effort required to develop and maintain any systems. With DSLs, reuse is feasible at model level, making it possible to reuse partial or entire models, rather than pieces of platform-specific code. Thus, the beginning of a new software development project can be done from existing reusable assets.

We have identified two key aspects that determine the feasibility of reuse in the context of DSLs: (1) in order to select a model or a model fragment for reuse, you must first know what it does; and (2) in order to have effective reuse, you must be able to discover the model fragment faster than you could build it from scratch.

The existence of a generic requirement set for subsequent system instantiation can significantly contribute to model reuse. Modelers could select a subset of these generic requirements when developing new systems. For each generic requirement a model fragment can be 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 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. HA domain is quite adequate to promote reuse by means of model fragments since requirements are well structured and it is quite straightforward to model them using the DSL.

However, users must be aware of that not every single developed artefact can be reused. But reusing is easier when working with models than it is when using code.

Lesson 3: Leveraging Existing Resources

It is essential to exploit the commercial tools available for each home automation technology rather than build a new set of tools from scratch, since it will facilitate the generation of executable code.

If we take the example of the KNX/EIB technology, at present, the starting point for a traditional developer is the creation of the project by using the tool ETS (Engineering Tool Software), which allows the creation of application code for this platform by means of a scripting language. With the proposed framework, the ETS tool becomes a support for the implementation of the code obtained. The ETS-ITTools plug-in allows us to interface the manufacturer environment using the VBScript programming language. The KNX specific model (level PSM) is the source for the model-to-code JET transformations to obtain the VBScript macros. These macros are then executed using the ITTools plug-in to automatically generate the project in ETS for any HA application.

This approach is interesting in the way that it takes advantage of the technology tools for code generation rather than building a whole new set of tools from scratch. In this way, the starting point for a traditional developer would be the creation of the project by interacting manually with the ETS tool.

Lesson 4: Benefits Obtained with Traceability

The incorporation of traceability in the MDA approach is a very interesting issue. A traceability management tool is necessary to have an automated way of visualizing the relationships between the artefacts obtained in the generation process, from requirements to code(Lago, 2009).

The framework described in ()includes a traceability management tool that generates reports from information collected in the traceability models created automatically during the execution of the modeltransformations steps. This report has been demonstrated of great interest to:

- Validate whether all the requirements have been supported. We have checked that all the requirements are represented by means of DSL model elements.
- Establish the impact of changing a requirement. It is usual that the stakeholders of the system suggest changes to the requirements throughout the whole development process. The traceability report has helped analysts to evaluate the impact of these changes before apply them.
- Verify whether the DSL model is compliant with the requirements.

Lesson 5: Eclipse Modelling Tools are Still Difficult to Use

Some difficulties have arisen at the creation of the DSL tools with Eclipse plugins (Eclipse Fundation, 2007) such as EMF, GMF and GEF. Although they are offered as intuitive graphical interfaces, rich in colors and shapes, speaking in terms of development they are not so easy to use. This is due to the great complexity involved in using the tools and the lack of adequate documentation.

EMF is the technological base on which to support other modeling tools. The naming convention of interfaces/classes generated by EMF plugin and the location of the generated files must be taken into account during the definition of the meta-model. For example, some words reserved by EMF can cause failure if selected as the name of a class, an interface or a reference.

GMF Eclipse plugin, used to generate graphical editors, tries to hide all the models by means of wizards. When you want to customize the generated editors it is needed to learn the use and origin of each of these models and GMF main functions in the development of the editor. A serious drawback of GMF is the lack of a graphical editor to create and manage the DSL elements. Besides, the definition of new visual elements as a composition of those previously defined is not a trivial issue. Thus, developers must define relationships among nodes and connectors manually.

The constraints should be defined and included in the design phase of the DSL. This fact has the drawback that any later change or addition of one of these OCL rules implies the rebuilding of the DSL editor.

These modeling tools are relatively young, they are still under development and new versions are released frequently, so often different versions are incompatible with each other. The documentation is rather basic and almost non-existent, often the best source of information can be found in the web news. It must also be highlighted the learning time needed to efficiently work with these tools (3-4 months).

3 CONCLUSIONS

Aframework has been created following the MDE approach which has allowed the development of a number of tools (requirements and traceability managers, DSLs, model transformations, and so on) to support the full development of HA systems. These resources have helped users to create HA applications in a way much more easier and productive.

The availability of tools to promote the creation of DSLs(Kelly, 2009) is quite promising. DSLs allow describing systems in an easy and intuitive way, using concepts from the application domain.

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.

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.

ACKNOWLEDGEMENTS

This work was partially supported by the Spanish CICYT project EXPLORE (TIN2009-08572).

REFERENCES

Eclipse Foundation, 2007. Platform Plugin Developer Guide, Platform Architectur. *http://help.eclipse.org/help33/index.jsp?topic=/org.eclipse.platform.doc.isv/guide/arch.htm.*

Echelon Corporation, LONWORKS Engineering Bulletins, 2009.

Sánchez, P. ,Jiménez, M., Rosique, F., Álvarez, B. and Iborra, A., 2011.A framework for developing home automation systems: From requirements to code. *Journal of Systems and Software*.88 (6): 1008-1021.

Karlheinz, F., 2009.EIB/KNX: Grundlagen Gebaudesystemtechnik. Edition MundArt, 4th ed.

Kelly, S. and Pohjonen, R., 2009.Worst Practices for Domain-Specific Modeling. *IEEE Software*. 26(4): 223-29.

Krueger, C., 1992.Software reuse. ACM *Computing Surveys*. 24(2): 131-183.

Lago, P., Muccini, H., Vliet, H., 2009. A scoped approach to traceability management. *Journal of Systems and Software*, 82(1), pages 168-182.

Mellor, S., Scoot, K., Uhl, A.and Weise, D., 2004. MDA Distilled*:* Principles of Model-Driven Arquitecture.Addison Wesley.

Miori, V., Tarrini, L., Manca, M. Tolomei, G., 2006. An open standard solution for domoticinteroperability. *IEEE Transactions on Consumer Electronics. 20(50), pages 19-25.*

Selic, B., 2003. The Pragmatics of Model-Driven Development, *IEEE Software*.20, pages 46–51.

Technica Pacifica, 2005. *Easy X10 Projects for Creating a Smart Home.*