Traceability Support for MDE Development of Home Automation Systems

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

Keywords: Traceability, Model-driven Development, Home Automation Systems.

Abstract: Traceability is a technique to ease determining the impact of changes in the design of software, to support their integration, to preserve knowledge, and to assure the quality and accuracy of the overall system. In this paper, an approach that considers traceability in the context of model-driven development of Home Automation (HA) systems is presented. This combination enables the development of tools with techniques for improving the quality both of the process and of the models obtained. To obtain these advantages we have developed a tool that provides users with traceability reports after applying model transformations. These reports enable developers to study whether all requirements have been considered, the impact of changes, and how they are considered both in architectural decisions and code implementations.

1 INTRODUCTION

The Model Driven Approach (MDE) (Selic, 2003), promotes the use of models as the main artefact in software development. A model is an abstract representation of reality that only shows those aspects that are of interest for a given purpose. A model is defined according to a meta-model that in turn defines the abstract syntax of a modelling language and establishes the concepts and the relationships among them. Since a system can be described by means of different models with different levels of abstraction, model transformations are one of the key issues of this approach (Mens and Van Gorp, 2006), since they encourage models to evolve towards other models that use concepts of a specific technology and in this way obtain executable code.

In a MDE process, traceability is crucial due to the extensive use of transformations (i.e. automated creation of artefacts). Traceability becomes central to be able to understand how and why an artefact was created (Kolovos et al, 2006). Traceability refers to "the ability to describe and follow the life of artefacts, in both a forward and backward direction" (Lagos et al., 2009) Forward traceability traces the software devices that are obtained in the normal development process. Backward traceability, on the other hand, aims to link each software element with the devices involved in its creation. The traceability of software artefacts offers much more detailed information on the adaptation of the developed system as well as the implications that any change may have. In (Behrens, 2007) Thomas Behrens defines several key goals of traceability in software development. We can summarize these capacities in the following items:

(i) To validate whether the different requirements have been taken into account.
(ii) To validate whether the obtained implementation complies with requirements and whether they have been satisfied.
(iii) To identify the impact that any requirements modification may have.

In this context, MDE transformations may record links (traces) between source and target elements. These traces can be useful in performing impact analysis, synchronization between models, model-based debugging and determining the source or target of a transformation. Traces make it easier to validate whether transformations are properly carried out, obtain support for an integrated management and evaluation of the impact of changes at different levels of abstraction (requirements, architectural design, detailed design and code).

The DSIE research group at the Universidad Politécnica de Cartagena has experience in the development of reactive software systems such as

Rosique F., Sánchez P., Alonso D. and Jiménez M..
Traceability Support for MDE Development of Home Automation Systems.
DOI: 10.5220/0004081302240229
In *Proceedings of the 7th International Conference on Software Paradigm Trends* (ICSOFT-2012), pages 224-229
ISBN: 978-989-8565-19-8

tele-operated service robots (Alonso et al., 2008), (Iborra et al., 2009), wireless sensors and actuators networks, and home automation (HA) (Jimenez et al., 2009). The proposal for HA systems development uses the Model-Driven Architecture (MDA) (Mellor et al, 2004) for organizing the software development in three layers: (1) a computation-independent model (CIM) represented by the syntax and part of the semantics of the defined DSL, (2) a platform-independent model (PIM) built as a component model and (3) a HA platform-specific model (PSM). The developer elicits requirements through the DSL in the CIM layer. Models from this level are automatically transformed into architectural components in the PIM layer. This level is a junction point for different reactive systems (wireless sensor networks, robotic systems, artificial vision, etc.). Consequently, the elements of HA systems designed in this manner can be integrated as components of a more complex reactive system. The tool then transforms the components into executable models for specific platforms.

The possibility to augment all the models involved in this process with traceability mechanisms has motivated the present paper. The rest of this paper is organized as follows: Section 2 introduces our previous experience with MDE and HA. Section 3 details the proposed framework for HA systems. Section 4 describes the integration of traceability in a MDE framework. Section 5 analyses other related papers and finally Section 6 presents the conclusions and further work.

2 TRACEABILITY SUPPORT FOR HOME AUTOMATION SYSTEMS

Figure 1 shows the proposed development framework of HA systems following the MDE approach, and its extension to include traceability. As can be seen on the right hand side, the different MDE levels corresponds to (1) HA requirements, (2) domain specific languages, (3) a component based level, and (4) executable code for a specific platform. Traceability support includes the artefacts shown in the dashed square on the left hand side of the figure. As can be seen in the figure, 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 should inspect this catalogue identifying the requirements that can be reused from previous applications. From the DSL level, a set of model transformation automatically generate (1) code implementation of the application (at the moment only for the KNX/EIB platform), and (2) a set of traceability models (left-hand side of Figure 1) of the whole process. These traceability models are later processed by TRT (*Trace Report Tool*) that provides the user with reports which are useful in different situations as will be described below.

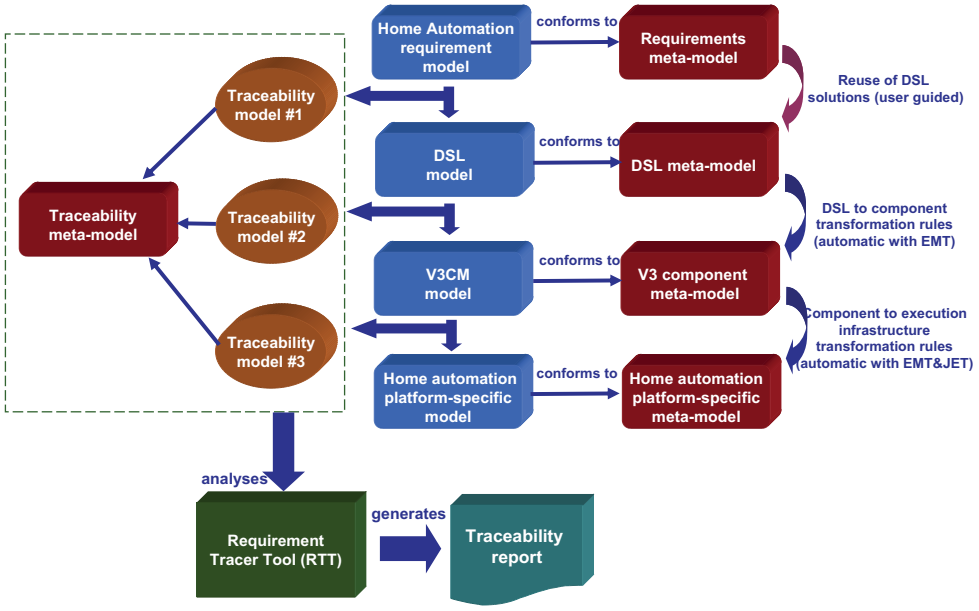


Figure 1: Framework for home automation model driven development.

A detailed description of each of the meta-models falls outside the aim of this paper. Nevertheless, the relevant details can be consulted in (Jimenez et al., 2009).

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 artefacts 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. The other is to keep the traceability information in an external model. This approach has the advantage of keeping models clean by facilitating loose couplings between models and links. The software development framework presented in this paper considers the second approach.

2.1 A Meta-model for Traceability

The traceability meta-model detailed in Figure 2 has been inspired by other works such as (Melby, 2007). This meta-model contains a *Link* 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 granulated levels 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.

The process takes into account a traceability model between each two consecutive levels of abstraction, as is shown in Figure 1. For example, in the particular case of the traceability between HA requirements and the DSL, the *Link* element is defined as:

- A *ModelElement* (*source*) which references an element of the HA requirements meta-model.
- A *ModelElement* (*target*) which references an element of the DSL.

Figure 3 shows all the possible combinations of traceability links supported by the meta-model between two consecutive levels of abstraction (for instance, from requirements to design, from design to elements of the programming language, etc.). As can be observed, it is possible to integrate the one-to-many relationships into one Link. At the same time, a target can be the destination of diverse traces with different origins.

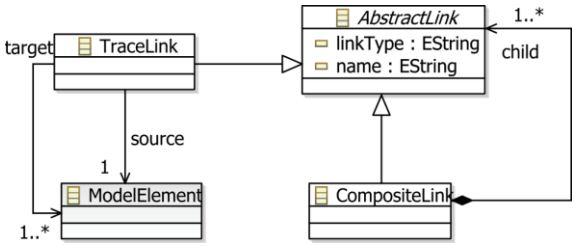


Figure 2: A meta-model for traceability.

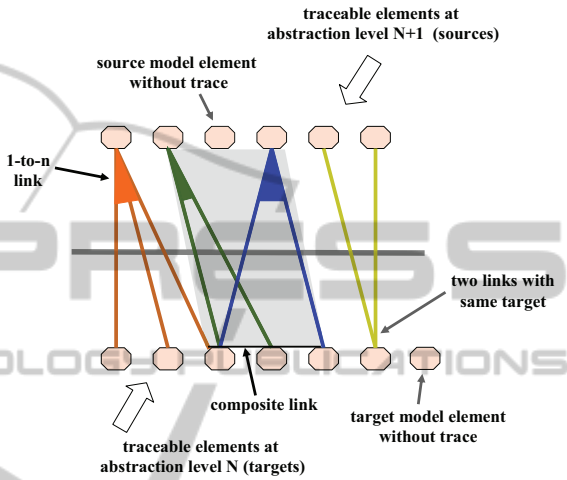
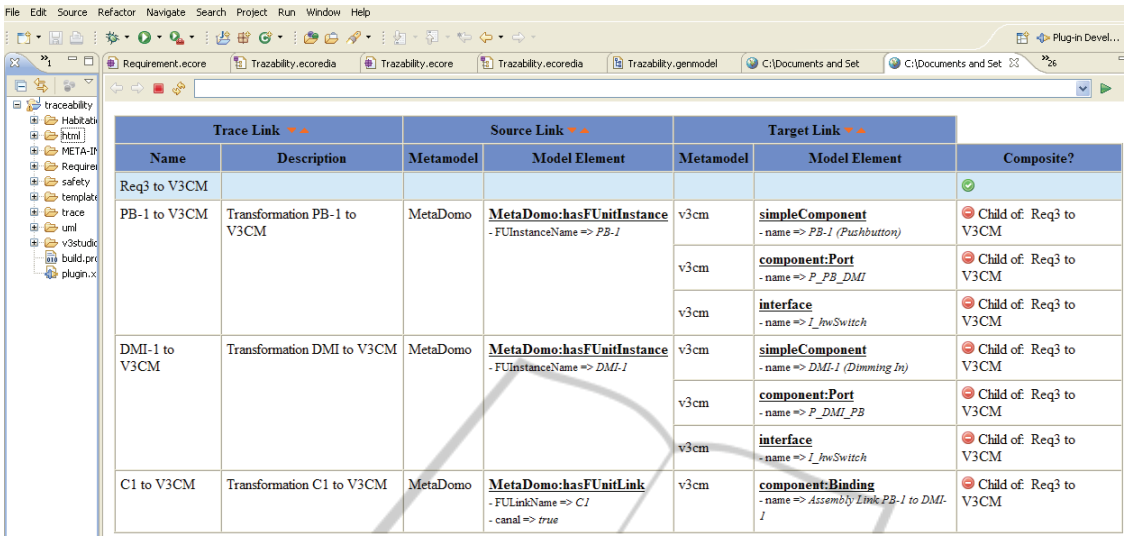


Figure 3: Examples of traces that can be represented with the proposed traceability meta-model.

2.2 Traces Generation

Trace links between source and target artefacts of a transformation may be created in implicit or explicit way. The first traceability model (between HA requirements and the corresponding DSL model) is created manually by the user. From the DSL until the final generated code, all the traces are obtained automatically as part of each of the model-to-model transformations. Transformations are defined using a graph grammar-based approach (Mens, 2006) (in particular the EMT plug-in for the Eclipse environment). The fact that models are usually represented as graphs makes graph grammars more suited than other approaches.

In order to populate the traceability model at the same time the transformation is being executed, the rules must be extended by generating a new *TraceLink* element each time a rule is executed. Each *TraceLink* collects the necessary information (name, description) for the Link type instances of the traceability meta-model as well as the matches between source and target.



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

Figure 4: Traceability report created by TRT: from Home Automation requirements to DSL.

2.3 The Trace Report Tool (TRT)

Several authors give recommendations about the issues that should be taken into account when designing a tool for managing traceability (Melby, 2007), (Oldevik and Neple, 2006), such as (1) to be able to manage models at different abstraction levels, (2) to consider the same meta-model for all the abstraction levels, (3) to store the traces in a persistent medium, (4) to be able to identify when a trace was created and the location of the referenced element(s), (5) to be able to integrate the tool with external applications, and (6) to be able to generate the traces both manually and automatically. With these issues in mind we have developed the *Trace Report Tool* (TRT) which allows developers to generate detailed reports from the trace models. Trace inspection can be used to retrieve all generated artefacts, the transformation responsible for creating them, analyse how a requirement has been taken into account in the process of automatic code generation as well as including information on the solution adopted. For each one of the traces, details of name and description are given, as well as information related to the source and the target of the traced elements.

Figure 4 shows the traceability report corresponding to the "Traceability Model #2" (see Figure 1). It is generated automatically as part of the transformation process from the DSL to the component-based model.

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. For instance, if an external light detector for the automatic power on of lights is added it will be necessary to: add new elements that might already be present in the system to satisfy other requirements 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.

3 RELATED WORK

The work presented here, as has been seen, support the traceability of software artefacts linked to HA requirements within a MDE framework. In this section the related work is analysed. This deals totally or partially with the three concepts that concern this paper: MDE, HA systems and traceability.

The literature offers few examples of work which tries to reach in an integrated way the development of HA systems using an MDE approach. Among these, it is important to highlight the works of (Muñoz and Pelechano, 2006) and (Voelter, 2007) that outline the necessity of using a MDE approach in HA systems. 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 MDE offers in the development of HA systems, but they also present some drawbacks. In the first place, J. Muñoz uses the UML notation for requirement capturing, which is not very intuitive for experts in the field of HA. In the work of M. Voelter it is necessary to build a new meta-model for each application using the Tree Editor tool provided by EMF the plug-in for Eclipse. Secondly, in both proposals the code generation is oriented to obtain OSGi (*Open Service Gateway Initiative*) components for a server or middleware platform (normally implemented in Java), and not to the programming of the HA devices. Therefore, it will always be necessary for an expert of the specific platform to program these devices. Contrary to the previous examples, in our environment the level of abstraction and usability of the requirement modelling rises with the use of a graphic DSL that uses specific concepts of the HA domain. In addition, our work guides the code generation to the automatic programming of the devices of the 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.

On the other hand, there are various papers that successfully approach traceability within the MDE framework (Winkler and von Pilgrim, 2010). (Melby, 2007) presents a traceability tool that is capable of defining and handling semantically rich traceability information in MDE, allowing traceability classification schemes to be generically defined, and which can be used to populate trace repositories with traceability information. Similar approaches are suggested in (Ramesh and Jarke, 2001), where a general purpose meta-model for requirements traceability that covers the most basic aspects of traceability is presented. These contributions coincide with the proposal presented in this paper, in which traceability models are obtained from the transformations integrated in an MDE environment. They also use a traceability tool that generates reports that allows the exploration of the elements of origin and destination of all the transformations carried out.

Contributions that deal with traceability of software devices for HA requirements within a MDE framework in an integrated way are hard to find. Markus Voelter (Voelter, 2007) proposes a generic Aspect-Oriented and Model-Driven Software Development (MDSD-AOSD) approach for product line implementation. This approach supports variability implementation, management and traceability throughout the development lifecycle. The paper presents a case study of the HA domain which demonstrates the viability of the methodology. In addition, techniques to incorporate the traceability and the prospective benefits of their application are presented. However, in terms of a generic approach, these ideas are not totally adaptable to all the requirements of HA applications. They also fail to explicitly validate traceability in concrete scenarios.

Our work differs from all the previous work in the following ways: (1) we provide a requirement meta-model that properly considers the HA requirements in the development of these applications; (2) we provide a model oriented framework to (semi) automatically derive implementations from HA requirements using a DSL; and (3) we provide a tool (TRT) that generates a traceability report from model transformations. With these results, the developer is offered an integrated environment which encourages the re-use of the software devices generated in the construction process of HA systems. In addition, by offering an environment for traceability, the quality of the built systems is increased, since it facilitates both modification as well as the impact analysis of the changes in the requirements.

4 CONCLUSIONS

The approach described in this paper focuses on tracing HA requirements to design, and design to implementation through different model transformations. We have shown the usefulness and value of combining a MDE approach with traceability capabilities when designing HA systems. This work represents a contribution that integrates in one solution the use of MDE and traceability in the development of HA systems.

The defined transformations represent the main artefacts on which the traceability relationships have been defined. By proceeding in this manner, the incorporation of traceability elements has been simplified notably between the source and target models, allowing the representation of this information in different traceability models that are generated throughout the whole process. In addition, a traceability report generated with the tool TRT is automatically obtained. This tool provides an interesting and flexible mechanism of inspection for the automation of traceability in MDE allowing (1) bringing together the whole information in a report, (2) supporting decision making throughout the whole process, and (3) checking the relationship among the software devices involved. With this

approach, it has been relatively simple to incorporate traceability throughout the whole development process since the meta-model used has dealt orthogonally and externally with the rest of the resources used.

The benefits of using the traces for concrete situations will represent a decrease in time and effort needed in the process. Although the case study this paper deals with is a simplified system, the results can be extrapolated to larger systems provided that certain improvements are carried out in the TRT. In this case, the main problem would be the necessity of managing a larger quantity of data. The possible inconveniences of this increase in data could be solved by adding to the tool advanced filtering features.

As further work, we plan to complete the model transformations for several HA platforms and to provide developers a user friendly framework which integrates all the involved tools in a single development system. In addition, work is currently underway to improve the TRT: to facilitate advanced filtering features, to extend the traceability framework with statistical information of elements generated in each level, to allow orphan analysis to find elements that are not the target of any trace link of a specific type (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, to allow the integration of the tool in environments other than Eclipse.

ACKNOWLEDGEMENTS

This work has been partially supported by the Spanish CICYT Project EXPLORE (ref. TIN2009-08572) and the Region of Murcia's Government Project MISSION-SICUVA (ref. 15374/PI/10).

REFERENCES

Alonso, D.; Vicente-Chicote, C. & Barais, O. "V3Studio: A Component-Based Architecture Modeling Language" 15th Annual IEEE International Conference and Workshop on Engineering of Computer Based Systems, IEEE, 2008, pp. 346-355, doi:10.1109/ECBS.2008.9

Behrens, T. 2007. Never Without a trace: Practical advice on implementing traceability, Available at: http:// www.ibm.com/developerworks/rational/library/feb07/ behrens.

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

Iborra, A., Alonso, D., Ortiz, F.; Franco, J.; Sánchez, P. & Álvarez, B. 2009. "Design of service robots" IEEE Robotics & Automation Magazine, Special Issue on Software Engineering for Robotics, , vol. 16, pp. 24-33, doi:10.1109/MRA.2008.931635

Jimenez, M., Rosique, F., Sánchez, P., Álvarez, and B., Iborra, A. 2009. Habitation: A Domain-Specific Language for Home Automation, *IEEE Software*, vol. 26(4), pp. 33-38.

Kolovos, D., Paige, R. And Polack, F.2006. On-demand merging of traceability links with models, in: *Proceedings of the 2nd EC-MDA Workshop on Traceability*.

Lago, P., Muccini, H., van Vliet, H. 2009. A scoped approach to traceability management. System and Software. vol 82 (1), pp. 168-182.

Mellor, S., Scott, K., Uhl, A., Weise, D. 2004. MDA Distilled. Object Technology. *1st ed., Addison-Wesley, Boston.*

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

Mens, T. and Van Gorp, P, 2006. A taxonomy of model transformation, *Electronic Notes in Theoretical Computer Science*. vol. 152 pp. 125–142.

Muñoz, J. and Pelechano, V. 2006. Implementing a Pervasive Meetings Room: A Model Driven Approach, in: *Proceeding of the 3rd International Workshop on Ubiquitous Computing*, pp.13-20.

Oldevik, J. and Neple, T.2006. Traceability in Model to Text Transformations, in: *2nd European Conference on Model-Driven Architecture Foundations an Applications (ECMDA'06)*.

Ramesh, B.and Jarke, M. 2001. Toward Reference Models for Requirements Traceability. *IEEE Transactions on software engineering*,vol. 27, nº 1, pp: 58 - 93.

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

Voelter, M. 2007. Product line implementation using aspect-oriented and model-driven software development, in*: Proceedings of the 11th International Software Product Line*, pp.233-242.

Winkler, S and von Pilgrim, J. 2010. "A survey of traceability in requirements engineering and model-driven development," *Software and Systems Modeling*, vol. 9, no. 4, pp. 529-569.