Automatic Impact Analysis of Software Architecture Migration on Model Driven Software Development

Joseba Agirre, Leire Etxeberria, and Goiuria Sagardui

Mondragon Unibertsitatea, MGEP, Mondragon , Spain
{jaagirre, letxeberria, gsagardui}@mondragon.edu

Abstract. The use of Model Driven Software Development (MDSD) approach is increasing in industry. MDSD approach raises the level of abstraction using models as main artifacts of software engineering processes. Models are closer to the problem domain than the solution domain and are easier to understand than the code. Models could be used for early validation and verification or for the automatic generation of code. When models are used for code generation, a system based on metamodels and transformations is developed in order to allow automatic code generation from models. Maintainability and evolution of these systems is a real and complex issue. Moreover, when the software architecture of the targeted systems evolves, the system that generates the code should evolve too. This means to adapt the transformation rules, the input metamodel and models.

To reduce the adaptation time of MDSD system has become crucial. In this paper we present an approach and a tool for performing automated analysis of the impact of software architecture changes due to evolution, concretely software architecture migrations, on model driven code generation systems. The approach and the tool improve the process of managing and implementing the required changes in MDSD due to Software architecture changes. To demonstrate the usefulness of the approach the tool has been applied to a MDSD system that generates ANSI-C code semi-automatically from UML models (a design based on UML2 components).

Keywords: Model Driven Software Development, Code generation, Impact analysis, Model to Model Transformation, Software Architecture Evolution

1 Introduction

Industrial software systems must continually evolve otherwise the solution they provide could become increasingly less satisfactory for the users and the marketplace. Concepts and techniques of Software Architecture are fundamental in software development [1]. A key aspect of software evolution is software architecture evolution. Depending on the requirements of the software evolution different abstraction level elements of the software architecture are affected. [2] Defines three kinds of software architecture evolution: Endogenous, architecture migration and exogenous migration.

Endogenous evolution refers to software architecture design refinements, such as splitting a component in two. Exogenous evolution refers to changes in the architecture modeling language. Software architecture migration occurs when the architecture must adopt a new technical infrastructure characteristic such as moving from client server architecture (CS) to service oriented architecture (SOA), or changing framework or execution platform. This work is focused on software architecture migration in which aspects, such as the concurrency API, the inter-component communication mechanism or execution platform evolve.

The combination of model driven software development (MDSD) and software architecture concepts is considered especially advantageous for developing complex systems. One of the weak points of MDSD is the management of the evolution of software architectures. When software architecture changes the adaptation of the designed and implemented MDSD system is critical. The aim of the work is to reduce the development cost of model driven code generation systems when a software architecture migration must be done. In this paper we present a methodology and a tool for automatic impact analysis of software architecture migrations in MDSD systems. The tool concretely deducts the transformation rules that must be modified and the changes that must be made in the transformation rules to adapt the MDSD system to software architecture migrations. The impact analysis tool is designed for MDSD systems that generate automatically code, but it can be used in any MDSD system that has a M2M transformation. First, in section 2, concepts of MDSD evolution and software architecture evolution concepts are explained. The third section describes the implementation of the impact analysis tool. Then in section 4 a case study which validates the usefulness of the tool is presented. Finally in section 5 and 6 we present the conclusions and future works respectively.

2 MDSD and Software Architecture Evolution

The adaptation process of MDSD systems for any kind of evolution has the same steps as in traditional software development :

(a) Identify an emergent need
(b) Understand and analyze the change impact
(c) Implement changes
(d) Validate the implementation

However in MDSD systems the evolution not only affects the final source code, also the metamodels, transformation rules (including templates) and models are affected. MDSD systems can evolve in different dimensions [3]: regular evolution, metamodel evolution, platform evolution and abstraction evolution.In regular evolution the models are the modified elements, the metamodels and transformation rules don't require any change. In metamodel evolution the modeling language is modified to increase its expressivity. The changes in the metamodel impact the models and therefore the models must be migrated to the evolved metamodel. In [4][5][9]

co-evolution between metamodels and models is treated .In metamodel evolution co-evolution between metamodels and transformation rules is also needed. In platform evolution, MDSD system must adapt the automatically generated output to new APIs, frameworks or provide different implementations of certain services. Such changes require modifications in the model to model (M2M) and model to text (M2T) transformations. When the source metamodel does not support the abstractions required for the new platform requirement it is necessary to extend the metamodel or to add a new metamodel. These situations are defined as abstraction evolutions and new domain concepts are inserted in the MDSD system. In abstraction evolution all the MDSD artifacts are affected.

Continuous adaptation of software architectures evolution in MDSD system is essential. Regarding evolution of software architectures, there are three different types of architectural evolution: Exogenous, endogenous and software architecture migrations. Table 1 shows the relation between architectural evolution and MDSD evolution.. Endogenous evolution requires only model refinements in a MDSD code generation system. Architectural exogenous evolution requires metamodels and meta-metamodels changes. . When architecture migration occurs the MDSD must adapt the transformation rules that generate the code and it may also be necessary to extend the architecture metamodel.

Table 1. SW Architecture Evolution relation with MDSD evolution dimensions

SW architecture Evolution type	**MDSD Evolution**	
	Dimension	*Elements to modify*
Endogenous	Regular Evolution	Models
Migration	Platform Evolution Abstract Evolution	Metamodels, models and transformation rules
Exogenous	Metamodel and meta-metamodel evolution	Metamodels, models and transformation rules

In most MDSD co-evolution proposals the input metamodel evolution differential is the initial entry point for the adaptation process. In these cases, changes impact analysis and changes implementation (co-evolution) requires adaptation mechanisms based on input meta-model evolution. [6] defines a framework based in megamodeling for analyzing the impact on transformation rules due to the evolution of the metamodel. The solution is based on relationships between elements of the metamodel and transformation rules for the impact analysis on the transformation rules. There are several works where different elements of MDSD are adapted automatically. For example, [4] and [5] automatically migrate existing models to new metamodel specifications. In [7] transformations are semi-automatically adapted based on the input metamodel evolution differential.However, sometimes the evolution requirement provides information about the changes to be made in the MDSD system output, for example in the

generated code. In these situations the input metamodel may require changes or not. For example, if a UML MARTE [8] design needs a change of the execution platform API, the metamodel is not affected. Sometimes changes in the input metamodel are also needed but it is not clear how to do the extension or there are several metamodel extension options. Architectural software often tend to create this kind of evolution scenarios in MDSD code generation systems. Even when the architectural evolution is expressed in the metamodel , as done in [2], predict and determine the changes of the transformation rules that generate the ouptut code requires an exhaustive inspect. In these situations an automatic impact analysis on the MDSD system transformation rules based on the output models is very useful both for designing a metamodel extension properly, as to guide the engineer in adapting transformation rules, especially when breaking and unresolvable changes[8] appears on the metamodel.

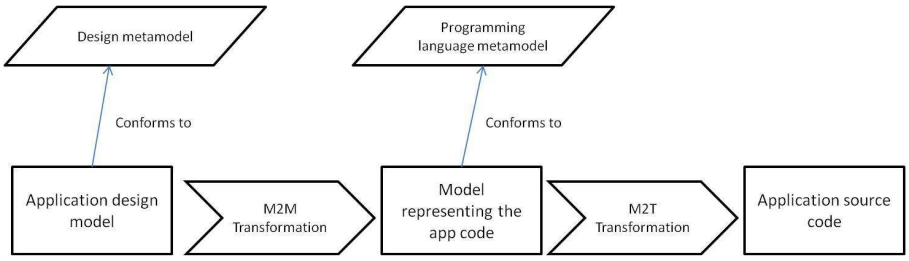

Fig. 1. MDSD code generation system generic design

In this paper, we present a tool for automatic impact analysis of MDSD code generation systems for SW architecture migration scenarios. The impact analysis is done on the M2M transformation. The solution uses a differential model of the models representing the code to establish the adaptations that must be made in the transformation rules. The differential model represents the SW architecture migration in the source level. The Figure 1 shows a MDSD code generation system with an intermediate step that generates a model that represents the code; the approach is for this kind of MDSD systems. The impact analysis process due to architectural software migration consists of the following phases:

1. Capture new software architecture migration requirement
2. Increase manually a previous output model without the requirement and obtain the differential model
3. Obtain the traceability between a previous design model, an output model without the requirement and the transformation rules
4. Deduct the adaptations to be made in the transformation rules of the MDSD system.

To automate the analysis process a JAVA & EMF tool has been created. The tool can be applied to any MDSD system implemented with EMF and ATL. The tool is independent of the metamodel used in the design and the output metamodel as long as they are based on the EMF meta-metamodel.

3 Automatic impact analysis tool design

The automatic impact analysis tool and the process used are described in detail in this section, see figure 2. First how the SW architecture migration requirement must be captured is explained. Then how the output models differential and the traceability model must be obtained is described. Finally the algorithm used to deduct the required changes in the transformation rules is introduced.

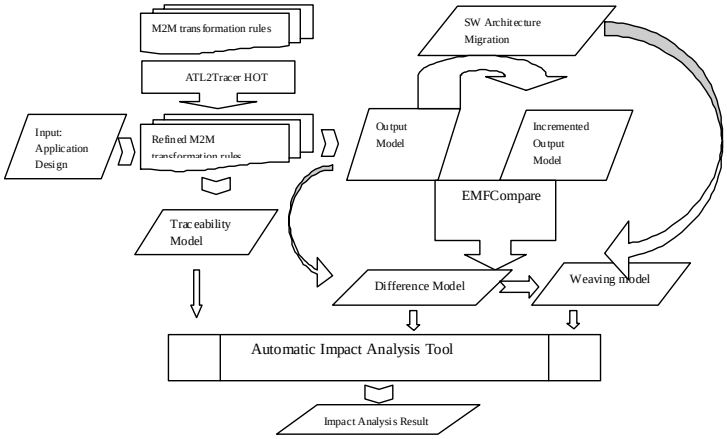

Fig. 2. Impact analysis process for architecture SW migrations

3.1 Capture new architectural software migration requirements

An EMF metamodel has been created to express the software architecture migration by actions to perform to achieve the requirement. This metamodel is based in [16] and allows specifying different level characteristics of source code that must be added and modified in the generated code when a software architecture evolution is needed. When a new software architecture migration requirement appears the software architecture engineer defines a model specifying the operations that are needed to adapt the software architecture of the applications. An architectural migration requirement example model is on figure 3. This metamodel can be replaced in the tool by another metamodel for expressing the output elements evolution without modifying the tool. This way the tool is independent of the metamodel used to specify the evolution, architectural or not architectural.

3.2 Output Models differential

Using the information of the architectural software migration requirement model the software architecture expert modifies a previously automatically generated output model to meet the new requirements. Knowledge on the generated software is needed in this step. Next the difference model between the incremented output model and the

previous model output is obtained. This differential is obtained by EMFCompare tool [10] that uses EMFDiff metamodel. After this step the software architecture engineer must establish the relationship between the differences and the architecture migration requirement model. This traceability information is used in the impact analysis algorithm. This relationship is done using Atlas Model Weaving (AMW) [11].

3.3 Transformation rules traceability

In the next step the ATL2Trace [12] Higher Order Transformation (HOT) [13] is applied to the M2M transformation of the MDSD system under development. The result of the HOT is a refinement of the ATL transformation rules under analysis. This new transformation rules creates traceability information when they are executed for an input design model and generates an output model that represent the source code of the design. The refined rules are executed with the input design model that creates the output model without new requirements. Each traceability data saves the target element, the source model element and the transformation rule responsible of generating the target model element.

3.4 MDSD system changes deduction phase

Using the differential model of the output models, traceability model and the weaving model between the architectural software migration requirement model and the output differential model the impact analysis can be done automatically. The tool performs the analysis in three stages: First it gets a complete list of possible changes to be made, then it performs a filtering process to avoid duplications and finally deducts the modifications to perform. The following sub sections describe the different stages.

Relationship between the output models differentials and transformations. First, it is established and collected which transformation rule is related to which difference of the output models differential. The tool gets the element of each output difference and searches in the trace model the transformation rule associated with the target element. Each relationship between a transformation rule and a difference is called adaptation goal. The tool relates EMFDiff metamodel difference with the modification operation to be applied on the transformation rules. Currently only *ModelElementChangeLeft* and *ReferenceChangeLeftTarget* difference types from EMFDiff metamodel are considered by the tool. For each *ReferenceChangeLeftTarget*, the type of modification that must be done in the transformation is to add a new binding. When the difference is a *ModelElementChangeLeft*, adaptation goal is established slightly differently. This difference type requires a binding in the related rule and a new rule to create the new output element. A simple metamodel has been defined to express adaptation goals. Adaptation goals are composed of: a source element, a target element, the transformation rule and rule refinement operations. The result data of the impact analysis is an adaptation goals model.

Adaptation goals filtering. Because the relationship between the transformation rules and the difference model is based on model elements and not on metamodel elements several adaptation goals may be referred to the same change to be made in the transformation rules. We therefore must filter adaptation goals to obtain the final list of changes. The filtering is based on the affected transformation rule, the source element type, the target element type and the software architecture migration requirement operation associated to the difference element. Due to space reason the filtering algorithm has been omitted.

Semi-Automatic adaptation of the transformation rules adaptations. With the information provided by the impact analysis, for example table 2 adaptations goals model, the MDSD system developers can start to make the required changes in the transformation rules. To facilitate this process a HOT has been defined to refine any ATL transformation rules using the impact analysis information. From the source and target elements the element type data is used. When an *addRule* change operation must be applied, a rule is created with its corresponding binding and is inserted into the corresponding ATL module. When the type of modification to be made in the transformation rules is an *addBinding* operation, the HOT refines this rule. The HOT also creates the header of the helper functions to be used in the binding assignments. For reason of space the HOT list is not shown, but if necessary you can request it via email.

4 Case study

In this section the automatic analysis of the impact is applied to a concrete MDSD code generation system. The MDSD system used in this case study can generate ANSI-C code for component-based SW architectures previously designed by UML [14]. UML designs are transformed to ANSI-C code through a M2M transformation and a M2T transformation, as shown in figure 1. In the first stage, the M2M transformation, the UML component models are transformed in SIMPLEC (a meta-model representing a subset of ANSI-C) models. M2M transformation is implemented in ATL. M2M transformation is performed incrementally by superimposition mechanism of ATL [15]. M2M transformation consists of 8 files, 40 matched rules, 30 lazy rules and 44 helper functions. In a second stage XPAND2 based templates are applied to SIMPLEC models to generate ANSI-C code. Two architectural software migration requirements have been used to test the impact analysis method and the developed tool. Originally the MDSD system of the case study did not offer concurrency characteristics on the components. The first migration requirement was to add concurrency capabilities to the generated code. The second requirement was a migration from a bare-metal cyclic executive concurrency API to freeRTOS. For space limitations only the first one is shown.

To perform the impact analysis a previously used UML design was selected: a UML design of an automatic door controller. To start with the impact analysis, first, the

architecture software migration was specified. The new platform requirement is to provide concurrent components under an API. Figure 3 shows the architectural SW migration model of this case study. The next step is to manually edit the SIMPLEC output model with the code elements necessary to use concurrent tasks under the selected API. Once the target output model with concurrency is created the difference model between the original and the incremented model is generated using EMFCompare Tool. A total of 12 changes were founded. In order to add information about the reason for the difference in the model representing the code, each difference is related to architectural software migration operations defined previously in software architecture migration model. Finally the traceability model is obtained.

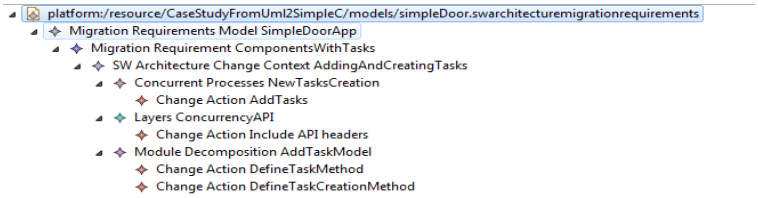

Fig. 3. Architectural software migration specification for adding concurrency characteristic

All the inputs needed by the impact analysis tool are ready: Traceability model, output differences model, weaving model between differences and software Architecture migration operations, the automatic door UML design and the corresponding door SIMPLEC model. With this information the tool can be executed. Table 2 shows the adaptation goal list for concurrency adaptation of the case study.

Table 2. Final adaptation goal list for concurrency adaptation.

Adaptation id.	Transformation rule	Source Element	Target Element	Modification operation type
1	lazy rule createAppComponentInstance	**Uml::Component** AppOS	**SimpleC::Module** AppOS	*addBinding*
2	lazy rule InstanceModuleFile	**Uml::Component** AppOS	**SimpleC::File** AppOS	*addBinding*
3	lazy rule createDeploymentPackage	**Uml::Component** AppOS	**SimpleC::Package** AppOS	addRule,*addBinding*
4	lazy rule componentInstance	**Uml::Component** SimpleDoor	**SimpleC::Module** SimpleDoor	*addBinding*
5	lazy rule InstanceModuleFile	**Uml::Component** SimpleDoor	**SimpleC::File** SimpleDoor	*addBinding*
6	rule createComponentPackage	• **Uml::Component** SimpleDoor	• **SimpleC::Package** SimpleDoor	addRule,*addBinding*
7	x	x	**SimpleC::Method** AddTask	addRule,*addBinding*
8	x	x	**SimpleC::Method** Schedule	addRule,*addBinding*

This architectural software migration requires platform evolution and abstract evolution in the MDSD system because requires changes in the generated code and new metamodeling elements. The OMG MARTE profile SRM package [8] was selected to specify the concurrency in the design. The changes implemented in the transformation rules to adapt the MDSD system to the new requirements were those suggested by the tool. This same process has been used successfully to adapt the selected MDSD system to a different concurrency API. Due to the design characteristic of MARTE pro-

file SRM package this architecture migration requirement did not need different or new metamodeling elements, so platform evolution was only required in the selected MDSD system.

5 Conclusions

The article has presented an impact analysis method for MDSD code generation systems for software architecture migrations. The analysis method has been automated by a Java tool & EMF. The benefits of using the tool have been also demonstrated comparing the impact analysis done without and with the tool for a selected MDSD system in the context of two software architecture migrations. The tool can be applied to any MDSD legacy system that has a M2M transformation implemented in ATL. To apply the tool it is enough knowing the changes that are necessary in the M2M transformation output models. The tool is independent of the metamodel used to express the evolution. In this case a metamodel has been defined to specify software architecture evolutions. There are studies about co-evolution for models migration [4] [5] and for adaptation of transformations [7] when metamodel evolution occurs. The work presented complements these works because it deducts changes that should be done in the transformation rules independently of the input metamodel evolution. In [6] mega-modeling is used to determine the impact that may raise evolution of a meta-model on the transformation rules. This type of solution requires previous knowledge of the MDSD system to establish the corresponding relationships between the meta-model elements and the transformation rules. Using only input and output models previously used to validate the MDSD system the presented tool can be used to understand the MDSD system and establish the relationships necessary for the mega-modeling. The work shows that combining traceability information and output models differential it is possible to analyze the impact of evolution requirements for M2M transformations.

6 Future work

The tool has been tested with one MDSD system case study. It is necessary to apply the automatic impact to other MDSD systems. Currently the tool only works with two types of EMFDiff difference types. The tool must be extended to deal with more difference types. Therefore, it is essential to analyze different types of software evolution and architecture migrations situations. This new situations will require new refinements operations and patterns for the transformation rules. At short term, the impact analysis result will be used in the design of metamodel extensions in software architecture migration situations that require abstraction evolution of the MDSD system. The goal is to predict the adaptation time of a MDSD system mixing the impact analysis data and the metamodel extension design.

7 Acknowledgments

This work has been developed in the DA2SEC project context funded by the Department of Education, Universities and Research of the Basque Government. The work has been developed by the embedded system group supported by the Department of Education, Universities and Research of the Basque Government.

References

1. Jazayeri M., “On architectural stability and evolution”, Proceeding of Ada-Europe’02, 2002
2. A. Amirat, A. Menasria and N. Gasmallah, "Evolution Framework for Software Architecture Using Graph Transformation Approach," The 2012 International Arab Conference on Information Technology (ACIT'2012), 2011.
3. A. van Deursen, E. Visser, and J. Warmer, Model-driven software evolution: A research agenda, in Proceedings of Int. Workshop on Model-Driven Software Evolution (MoDSE) held with the ECSMR'07, March 2007
4. Herrmannsdoerfer, M., Benz, S., Juergens, E.: COPE - automating coupled evolution of metamodels and models. In: Drossopoulou, S. (ed.) ECOOP 2009. LNCS, vol. 5653, pp. 52–76. Springer, Heidelberg (2009)
5. Louis M. Rose, Dimitrios S. Kolovos, Richard F. Paige, Fiona A. C. Polack: Model Migration with Epsilon Flock. ICMT 2010: 184-198
6. Ludovico Iovino, Alfonso Pierantonio, Ivano Malavolta: On the Impact Significance of Metamodel Evolution in MDE. Journal of Object Technology 11(3): 3: 1-33 (2012)
7. Jokin García, Oscar Díaz, Maider Azanza: Model Transformation Co-evolution: A Semi-automatic Approach. SLE 2012: 144-163
8. OMG: Modeling and Analysis of Real-time and Embedded systems (MARTE), Version 1.0, http://www.omg.org/spec/MARTE/1.0/. (2009)
9. Antonio Cicchetti, Davide Di Ruscio, Romina Eramo, Alfonso Pierantonio: Automating Co-evolution in Model-Driven Engineering. EDOC 2008: 222-231
10. A. Toulmé. Presentation of EMF Compare Utility. Eclipse Modeling Sympossium 2006, pages 1-8, 2006.
11. Didonet Del Fabro, M ; Bézivin, J. ; Valduriez, P . Weaving Models with the Eclipse AMW plugin. In: Eclipse Summit Europe 2006, 2006, Esslingen. Eclipse Modeling Symposium, Eclipse Summit Europe 2006, 2006
12. Joault, F., “Loosely Coupled Traceability for ATL”, University of Nantes, ATLAS - INRIA Group, 2005
13. Massimo Tisi, Jordi Cabot, Frédéric Jouault: Improving Higher-Order Transformations Support in ATL. ICMT 2010: 215-229
14. Joseba Andoni Agirre, Sagardui Goiuria , Leire Etxeberria."A flexible model driven software development process for component based embedded control systems", III Jornadas de Computación Empotradas JCE, SARTECO, Elche,Spain, 2012
15. Dennis Wagelaar, Ragnhild Van Der Straeten and Dirk Deridder, Module superimposition: a composition technique for rule-based model transformation languages, Software and Systems Modeling, 2009
16. Byron J. Williams, Jeffrey C. Carver: Characterizing software architecture changes: A systematic review. Information & Software Technology 52(1): 31-51 (2010)