ModifRoundtrip: A Model-Based tool to reuse legacy transformations

Paola Vallejo¹, Jean-Philippe Babau², and Mickaël Kerboeuf²

¹ Universidad EAFIT
² Université de Bretagne Occidentale, Lab-STICC, CNRS, UMR6285 pvallej3@eafit.edu.co, {babau,kerboeuf}@univ-brest.fr

Abstract. The legacy transformations dealing with domain specific data gathers important expertise. Nevertheless, in many cases, have to be rewritten in order to make them apply to semantically equivalent but structurally incompatible data. According to the complexity of the transformation, rewriting them can quickly become a difficult and error-prone task. We propose a coevolution approach to enable the reuse of legacy transformations instead of their rewriting. In this approach, the data conforming to the reuse context coevolve into data conforming to the legacy transformation context. Legacy transformation is applied and the result is migrated back to the reuse context.

In this paper, we introduce ModifRoundtrip, a plug-in for the Eclipse development environment. It promotes the reuse of legacy transformations, providing guidance for the user during reuse process. Reuse process is done automatically, but ModifRoundtrip provides interaction points to the user to indicate custom reuse scenarios. A tool demonstration video is available at: https://youtu.be/iAPul2httyE.

Keywords: Coevolution, DSML, Legacy transformation, Reuse.

1 Introduction

Domain Specific Modeling Languages (DSML) support high-level abstractions and require few low-level details to specify a given domain. They are usually instrumented with dedicated transformations. Some of them are generic, some others are domain specific. In both cases, there are variants of the transformations that have already been implemented in other domains which are structural variants sharing common concepts. Based on those observations, when a transformation for a DSML is needed, instead of writing it, the reuse of a legacy transformation represents a promising way to build safely and quickly the equipment of a DSML [1]. However, even if legacy transformations are available, reusing them within the scope of a given DSML raises two problems: the adaptation of DSML data structures (user oriented) into legacy transformation data structures (computation oriented); and the extraction of a subset of relevant concepts for the targeted transformation, from the large set of concepts defining the DSML.

2 Reusing legacy transformations with ModifRoundtrip

As data handled by the transformation to be reused and data handled by the DSML are structurally different, the reuse of the transformation implies to adapt DSML models in order to put them under the scope of the transformation. And, more important, to adapt the resulting models to put them under the DSML scope. Making tools facilitating transformations in both directions is essential.

In this paper we present ModifRoundtrip [2], an Eclipse plug-in that promotes the reuse of legacy transformations, reducing manipulation of models. The remainder is structured as follows: Section 2 presents related work concerning model coevolution as a way to adapt models. Highlights of the tool are presented in section 3. Section 4 briefly describes the architecture of the tool. Finally, section 5 presents the conclusions and the ongoing work.

2 Background

In this paper, we assume that all the data that are required by the targeted transformation are available in the DSML even if they are structured differently. Considering this condition, we use the principles of metamodel evolution and model coevolution to automate the navigation between semantically close DSMLs by means of model migrations. ModifRoundtrip operation relies on these principles [3] (fig. 1). *Refactoring* and *Migration* keep and update a subset of the concepts of the DSML. ModifRoundtrip produces a migrated model (mm) from the initial model (im); then the transformation produces the result (pm). Finally, *Reverse Migration* migrates back it into the DSML context (rm). The elements that had been removed during *Migration* are automatically and as much as possible recovered without undoing the transformation's action. Elements created by the transformation are also integrated with initial elements.

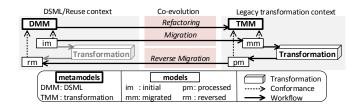


Fig. 1. Model migrations for the reuse of a legacy transformation

With ModifRoundtrip, the user indicates the coevolution operators (remove, rename, hide, etc.) to be applied to the DSML in order to make it match the transformation metamodel. Using this operators does not require specific knowledge about model transformations. The migration in Epsilon Flock [4] integrates a conservative copying algorithm and user-defined migration rules. EMFMigrate concentrates on metamodel evolution and coevolution, however it is used by [5] in the frame of metamodel evolution and transformations coevolution. Bento [6] supports the development and the execution of reusable transformations, based on the use of generic templates. Genericity [7] increases the level of abstraction by defining metamodels containing the minimal requirements for applying a transformation. transML [8] allows the development of model transformations following a MDE approach (transformations are viewed as models). MTBE [9] and MTBD [10] propose solutions for simplifying the implementation of model transformations by using a visual support. Edapt³ coevolve models after metamodel transformations by means of customizable migration code. ModifRoundtrip aims at reusing existing transformations, adapting the models but the transformations. The distinguished feature of ModifRoundtrip is that, unlike the methods described above, our tool produces editable migration specifications, avoiding the migration code completion by the user. This become an usable solution for non-experts in model migration or programming languages. As the SyVOLT tool [11], ModifRoundtrip hides formal details from the user and place essential user interactions at the beginning of the reuse roundtrip.

3 Highlights

ModifRoundtrip is released as an open source software. It relies on the Eclipse Modeling Framework (EMF) and Xtext⁴ for defining the modif models and for realizing refactoring specification editors. Git⁵ is used as version control system. The plug-in and source code of ModifRoundtrip can be found at http: //pagesperso.univ-brest.fr/~vallejoco/Modif/pages/indexModif.html.

Integration with Eclipse. ModifRoundtrip takes advantage of the EMF ecosystem to specify and validate models. Users can stay within the Eclipse environment, when they develop the tooling for a specific domain. ModifRoundtrip provides an Eclipse Perspective, containing three views and two buttons (fig. 2). Views allow to edit specifications and visualize the current state of the reuse process. Buttons allow to create projects and to start the reuse process.

Formalization. The ModifRoundtrip's reuse process formal details are hidden from the user. The tool behavior is formally defined in [12]. Once the differences between the DSML and the transformation metamodel are specified by the user, the roundtrip reuse process is started. It automatically creates all required artifacts and provides the result to the user within the Eclipse environment.

Input metamodels independence. The metamodels are designed by using Ecore (EMF). ModifRoundtrip operation is not dependent on specific input metamodels. Nevertheles, at now, the legacy transformation to be reused must be written with the Java language respecting the generated EMF interface.

Customizable reuse. Obtaining processed models according to specific user needs, depends on the input model given to the legacy transformation. ModifRoundtrip provides editable migration specifications. Allowing users to produce different input models for the reused transformation, from the same migration specification. User can also apply the tool to portions of the model.

³ https://www.eclipse.org/edapt/

⁴ https://eclipse.org/modeling/emf/ and https://eclipse.org/Xtext/

 $^{^{5}}$ https://github.com/

4 Reusing legacy transformations with ModifRoundtrip



Fig. 2. Eclipse plug-in of ModifRoundtrip

Traceability. ModifRoundtrip identifies model instances by means of Universally Unique IDentifiers (UUID), it allows to trace instances evolution through the reuse process. Furthermore, instances created by the reused transformation can be associated with initial model instances.

Scalability. ModifRoundtrip has been used to reuse transformations in the real and complex Orcc context, a compiler for dataflow applications defined by 70 classes, 100 references and 50 attributes. It underlined how the development of the transformations for a given DSML can be improved by using our tool [3]. It can be tested on other domains regardless of the complexity of the metamodel.

GUI or API. ModifRoundtrip can used by means of a Graphical User Interface (GUI) or by calling the functions defined in the provided ModifRoundtripAPI. ModifRoundtrip can be used to perform separately: Refactoring, Migration or Coevolution. The ModifRoundtripAPI provides also functions to create, load and save models and refactoring specifications.

4 Architecture

The ModifRoundtrip Eclipse plug-in is divided into three components: 1) The User Interface, primary based on the Rich Client Platform (RCP) provided by Eclipse. 2) The modif specification editor, defined by means of Xtext. 3) The reuse process behavior, implemented by using Java and the EMF facilities. The reuse process comprises the modules (1) to (10) in figure 3. Squares annotated with numbers represent computation, squares annotated with letters represent produced and consumed data and squares with dotted lines represent user interactions. ModifRoundtrip's modules are orchestrated by a Java program. This program makes sure all components communicate and execute in the right order, and allows roundtrip to run fully automatically at the push of a button. We have used available model-driven development technologies as much as possible, both to develop and as a part of ModifRoundtrip itself. In what follows, we briefly visit each ModifRoundtrip computational component to describe its function.

(1) Input definition. The GUI is started by clicking the *Run* button of the ModifRoundtrip's plug-in. It allows user to specify the elements needed to reuse a legacy transformation. (a) points the metamodel describing the transformation

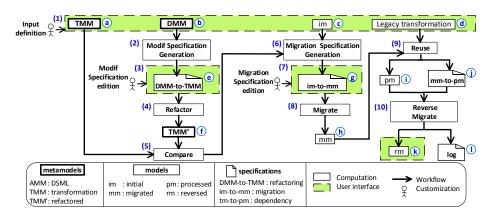


Fig. 3. Architecture of the ModifRoundtrip tool

to be reused, (b) represents the metamodel describing the reuse context (DSML in which transformation will be reused). (c) is the input models and (d) is the transformation to be reused. Figure 4 illustrates the GUI. Figure 5 illustrates an example of the use of ModifRoundtrip by means of the API.

New Finish			
	UML UML	NoModif @ EraseAll	modifRoundtrip =
Project source folder	DemoCoffeeMaker Select	Generate Modif Specification	new ModifRoundtrip <dmm, tmm=""></dmm,>
Modif Specification	C:\ModifRoundtrip-project\workspaceMR\De Select		EObject rm =
Tool Metamodel	C:\ModifRoundtrip-project\workspaceMR\Fla	Check	<pre>modifRoundtrip.reuseTransfo(im, DMM, TMM, DMM_to_TMM,</pre>
RootClass	C:\ModifRoundtrip-project\workspaceMR\Fla		TMM_, className, transfo,
Function	flatten		autoMigration,
Domain Model	C:\ModifRoundtrip-project\workspaceMR\co	Migrate	
Re	use Reverse Migration Clos	;e	serialize);

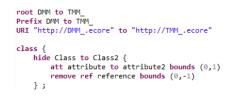
Fig. 4. ModifRoundtrip GUI

Fig. 5. ModifRoundtripAPI

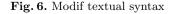
(2) Modif Specification Generation. We have developed a language for specifying the differences between the two metamodels of the input. A basis Modif specification template (a) is automatically generated, then the user indicates how to refactor the DSML in order to make it match the transformation. It is done by using the proposed refactoring operators.

(3) Modif Specification Edition. In the modif specification, the user indicates only the operators needed to transform the DSML into the transformation metamodel. As we consider transformation metamodel has only dedicated concepts, and adding elements is not needed, the used operators are basically: remove, rename, hide and flatten. Modif specification editor facilitates the edition of modif specifications by highlighting key words, checking syntax and suggesting the operators to applied. Figure 6 depicts an example of the modif syntax.

Reusing legacy transformations with ModifRoundtrip



6



#	EClass	Object	Remove	^
2	StateMachine	name : model	False	
1	Model	name : model	False	
3	Region	name : RegionBrew	False	E
7	Region	name : RegionLight	True	
			False	Ľ

Fig. 7. Migration editor

(4) **Refactor and (5) Compare.** Refactor takes the modif specification (e) and apply it to the DSML (b). It produces a refactored metamodel (f). Then Compare, verifies that (f) fully match with (a), if that's the case, the reuse process will start. If they do not entirely match, ModifRoundtrip notifies the user about where the matching's violation occurs (at now, in terms of deletion and rename). Then, the user edits the modif specification and Refactor is executed.

(6) Migration specification Generation and (7) edition. Migration specification (2) is automatically generated from the modif specification and the initial model. Technical details of the translation from modif specification to migration specification can be found in [13]. By keeping in mind the goal of reducing the skills required to define custom model migrations, we developed a user interface (Fig. 7) allowing the user to indicate specific modifications. If not customization is defined, the migrations is applied automatically. The user may act over its elements to indicate additional deletion.

(8) Migrate. It migrates the input model © into a model (1) in the context of the legacy transformation. Migration depends on the migration specification and it is performed by a model independent engine implemented in Java. This engine handles migrations in which one element of the migrated model is produced from one and only one element of the initial model.

(9) **Reuse.** The transformation is reused without user intervention, it takes the migrated model (h) as input and it produces a processed model (i) along with a dependencies graph (j). The dependencies graph keeps a trace about how the elements are modified or created by the transformation.

(10) Reverse Migrate. It applies the reverse operators of the migration specification operators. Reverse of *rename* is applied on the processed model (1) in order to produce the reversed model (2). Then, reverse of complex operators as *remove* and *hide* are applied on (3), in order to extended it with the elements of the initial model (2) that have been removed. It is based on the *natural join* concept of data bases. Then, each recovered element is related to its associated elements (as in the initial model). If the associated elements are not in (2), then, the recovered element is related to the elements that were created or modified from the associated elements. This information is provided by the dependencies graph. At the end, the model (3) is validated in order to respect the type

constraints imposed by the metamodel (a). This activity is automatic and transparent for the user if the migration does not involve creation, merge or instances division. The result model (k) keeps the modifications provided by the reused transformation, integrating initially removed elements and respecting the structural constraints of the DSML (b) in which the transformation is reused. The user is notified by means of a log file (1) about those elements that could not be included into the final result, so it can evaluate its pertinence and to add them in order to produce different results.

5 Conclusion and future work

This article presents a MDE-based tool for reusing legacy transformations. It provides a user interface that allows the user to deal with context-related concepts hiding all the intermediate models and transformations involved. The main motivation of this tool is to provide a methodology and a tooling set to assist users to reusing legacy transformations. Accordingly, we provide a modeling language that allow to express the modifications (to be applied) at a convenient level of abstraction and to interface with modeling environments. This tool allows to separate concerns of representation from core data semantics. In future work we aim at providing more operators and facilities to reuse legacy transformations defined in other languages.

References

- 1. Frakes, W. et al.: Software Reuse: Metrics and Models. ACM Comput. Surv. (1996)
- $2. \ Modif, {\tt http://pagesperso.univ-brest.fr/~vallejoco/Modif/pages/indexModif.html}$
- 3. Vallejo, P. et al.: Improving Reuse by means of Asymmetrical Model Migrations: An Application to the Orcc Case Study. In: MODELS 2015 (2015)
- Rose, L. et al.: Epsilon Flock: a model migration language. In: Software and System Modeling 13(2) (2014)
- 5. Di Ruscio, D. et al.: A methodological approach for the coupled evolution of metamodels and ATL transformations. In: ICMT2013 (2013)
- 6. Cuadrado, J. et al.: Reusable Model Transformation Components with bentõ. In: ICMT2015 (2015)
- 7. Rose, L. et al.: Genericity for model management operations. In: Software and System Modeling 12(1) (2013)
- 8. Guerra, E. et al.: Engineering model transformations with transML. In: Software and System Modeling 12(3) (2013)
- 9. Varró, D.: Model Transformation by Example. In: MODELS 2006 (2006)
- 10. Sun, Y. et al.: Model Transformation by Demonstration. In: MODELS 2009 (2009)
- Lúcio L. et al.: SyVOLT: Full Model Transformation Verification Using Contracts. In: CEUR Workshop Proceedings, P&D@MoDELS (2015)
- Kerboeuf, M. et al.: Formal framework of recontextualization by means of dependency graphs. Tech. report (2015)
- 13. Vallejo, P.: Reuse of legacy code for DSML tools in the context of MPSoC. PhD thesis, Université de Bretagne Occidentale (2015)